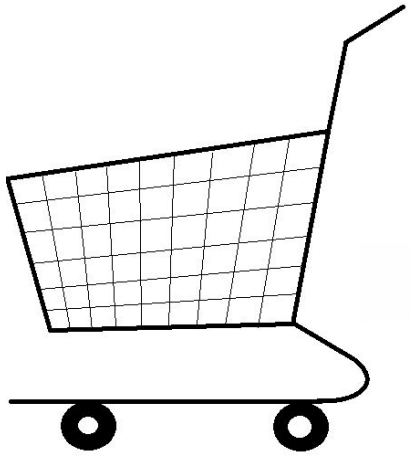


Automated Supermarket Report 3

Group 4

github.com/as2580/SoftwareEngineeringGroup4

Section 0: Report Documentation

Section 0.0: Table of Contents

Section 0.1: Individual Contributions Breakdown

	Andrew Saengtawesin	Kimberly Chang	Jake Totland	Andrew Vincent	Amali Delauney	Harsh Desai	Abhinandan Vellanki
Summary of Changes	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
Section 1: Customer Statement of Requirements	50.00%	50.00%					
Section 2: Glossary of Terms	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
Section 3: System Requirements	50.00%	50.00%					
Section 4: Functional Requirements Specification	5.00%	5.00%			30.00%	60.00%	
Section 5: Effort Estimation			50.00%	50.00%			
Section 6: Domain Analysis				33.33%	33.33%		33.33%
Section 7: Interaction Diagrams and Design Patterns					33.33%	33.33%	33.33%
Section 8: Class Diagram, Interface Specification, and OCL Contract Specification			50.00%	50.00%			
Section 9: System Architecture and System Design	50.00%	50.00%					
Section 10: Algorithms and Data Structures			50.00%	50.00%			
Section 11: User Interface Design and Implementation			50.00%	50.00%			
Section 12: Design of Tests			50.00%	50.00%			
Section 13: History of Work, Current Status, and Future Work	50.00%	50.00%					
Project Management	50.00%	50.00%					

Section 0.2: Summary of Changes

In Section 1.1, further elaboration is added for task creation regarding items in the returning items baskets, and alterations are made in description of the database.

In Section 1.3, the process of the Returns System is slightly modified in order to not have an employee present until the system calls for one. This is a more resourceful use of the employees. This change is also indicated in Sections 4.3, 4.4, and 7.

In Section 1.5, the cost of RFID readers was added. Additionally, the time cost for some tasks was added.

In Section 2, the definition for the term Customer Assistance Terminal was altered to reflect that the Customer Assistance Terminal runs a different program than the Customer Application.

In Section 3.1, the Customer Assistance Terminal and Customer Application requirements were separated to reflect that they are different programs. New requirements were made for registration of new employees, managers, and customers and for login functionality at Checkout Terminals

In Section 3.3, the diagrams were updated to be more readable.

In Section 7, added design patterns that help us organize our program much better.

Section 8.2 was edited to better reflect the actual program.

Sections 8.3 and 8.4 were added.

In Section 9.1, edits were made to show the architectural style of the current product based on the actual programming done.

In Section 9.4, information on generation for the sample database tables was added.

In Section 10, elaborated on how the linked lists were used and why they were chosen.

Section 1: Customer Statement of Requirements

Supermarkets can be chaotic places with all the customers scurrying around looking for their items and workers trying to keep stock on shelves. A disorganized and poorly managed supermarket can alienate customers and cause business to move elsewhere. In the supermarket industry, ensuring customer satisfaction is of utmost importance.

In the current market, adaptation to the usage of new technologies is essential. Large companies such as Walmart and Amazon are becoming the dominant players in the industry, so supermarkets must have competing technologies. The supermarket Earth Flare has already been forced out of business by these behemoths. [1] Walmart has a pick up in store option for customers, and Amazon has its own physical store Amazon Go automated without any employees.

It is imperative to implement technologies in a supermarket to increase the convenience of the shopping experience to better ensure customer satisfaction and retention, increase worker productivity, and reduce redundancy and inefficiencies.

Section 1.1: Item Management

Often, customers will be unable to find a specific item and must then search the store to find an employee to help them find the item. Shelves may be left empty until a customer asks an employee if there is any stock in the backroom. Items may simply be out of stock in the store, and more must be ordered from the supplier. Being unable to locate items could cause great dissatisfaction in customers, so managing stock is important for the maintenance of a supermarket.

Often, customers will place items on the wrong shelf because doing so is more convenient than returning the item to the proper shelf. This can cause disorganization of the inventory on the shelves, make finding items much harder for other customers, and create unnecessary tasks that must be handled by the employees. Therefore, making the process of removing unwanted items from a shopping cart should be made more convenient.

Expired items may be left on shelves, causing serious potential liabilities. Customers should not ever be able to add expired items to their shopping cart. The potential lawsuits of such a small oversight could cost a great amount for the supermarket.

Solution

Each product shall have an RFID, and each item will have its own RFID tag to allow for easier item scanning. Entire stores can be run without employees using items with RFID tags.[2]

At the end of each aisle, there will be a Customer Assistance Terminal. The Customer Assistance Terminal will be attached to the wall instead of a price checker; it will not be removable by the customer in order to prevent theft. The Customer Assistance Terminal will allow the customer to (1) check the location(s) of an item or items for which they are searching or (2) check the price of an item. When checking the location of an item, the customer will be shown a map of the store, and the aisle (ex. Aisle 7) or area (ex. Produce) of the store in which the item is located will be highlighted. The Customer Application will also be accessible on the customer's smart device in order to improve convenience and encourage customer retention.

There will be a basket for customers to leave an item to be reshelved. The RFID scanner inside the basket will constantly check to see if items are in the basket, and if an item is inside, a task will be generated in the Employee Application, and an alert will be sent to employees. The task is immediately generated instead of waiting for the basket to be full because there are items that may spoil (e.g. milk) if left out too long, and the basket may not be filled until sometime after the item is spoiled or even after the store closes. Therefore, the immediate creation of the task will ensure that items do not spoil and that items are not left in baskets for multiple days. This will improve the convenience of removing unwanted items from shopping carts and improve store item organization.

All items shall be contained in a database. There shall be a stock table in the database for the items on shelves and for the items in the back room. Each entry in the stock table shall contain a product's name. the number of that product that is on the shelves, the location of that product, its RFID, and its expiration date (if applicable). For items with expiration dates, they will be grouped in bundles (as most products are bought in bulk), and each bundle of items will be grouped together with the earliest expiration date. For more information, please see Section 9.4.

Section 1.2: Task Synchronization

Multiple employees may be performing the same task, resulting in redundancy and inefficiency. This could cause tasks to pile up and force employees to complete the plethora of tasks that need to be completed. Employees may also find it difficult to determine what tasks need to be done without manually monitoring the store to discover that an issue needs to be resolved.

Efficiency is key to running a supermarket smoothly. Employee tasks should be synchronized and organized in order to allow employees to perform their work more efficiently. Disorganized

employees will not be able to help customers smoothly and may cause the supermarket to appear substandard.

Solution
There shall be an Employee Application that will create tasks that employees can accept and complete. Tasks will be created for when a customer has placed an item to be reshelved in the Customer Assistance Terminal basket, when a customer has requested assistance, when items have past their expiration date and should be thrown out, and when items need to be restocked (in either the store or the back room). An employee can claim a task, but multiple employees cannot claim the same task. When an employee claims a task, the task will be marked as having been assigned. When an employee completes a task, they will mark the task as complete, and the task will be removed.

Employees will be able to clock in and out on the Employee Application to determine employee hours more easily. Managers will be able to view and correct employee hours on the Manager Application. If an employee requires assistance, they will be able to request help from a manager.

Section 1.3: Self-Checkout and Returns

Many supermarkets have a cashier to manually checkout customers. This can be inefficient because at times, a cashier may get no customers, and they will not have any work to do. Moreover, the speed of a customer's checkout relies on the speed of the cashier, and if the cashier is slow, this could cause an irritating and unpleasant experience for the customer, especially if they have been waiting in a line for a long time. Self-checkouts can help with this issue. Although some supermarkets have self-checkouts, they are still moderated by employees; it would be better if this system could be completely automated.

Returns can also be an irritating situation, and the process should be better automated. The process should still be moderated by an employee because a customer could potentially return an empty box, bag, etc. that may be detected by video as being a returnable item. However, expediting this process will improve efficiency and improve the customer experience.

Solution
Checkouts will be completely automated. There shall be Checkout Terminals by the exit. A customer will be able to use the Checkout Terminal to checkout their items. A customer can bag several items and leave them in the bagging area. The customer can then use the application on the Checkout Terminal to determine which items were bagged. The customer may also remove

that set of items from the bagging area and begin the process with the next set of items. If a customer requires assistance, they can request it on the application at the Checkout Terminal.

There will be a basket for items that a customer decides that they do not want. This will allow customers to easily remove unwanted items from their shopping cart without buying the item and without leaving the unbought item at the Checkout Area with employees unaware of the issue.

There will be a Returns Terminal where customers can return items that have already been purchased. There shall be an RFID scanning basket at the terminal that can determine the item to be returned. An employee will need to be called to ensure that faulty items are not returned. Post verification, the employee can hand the system back to the customer who will complete the refund of payment. The employee can then proceed to create a task to reshelve items in the return bin.

Section 1.4: Management

Supermarkets will always have employee turnaround. Many low-level employees only work there temporarily in preparation for other opportunities. Therefore, training should be minimized so as to improve efficiency.

Moreover, training excellent managers can be a difficult and time-consuming task, which includes finding the right people and providing enough training to allow them to reach their full potential in a managerial position. There will be many more low-level employees than there will be managers, and even the best managers can be overwhelmed by having to supervise so many workers. Furthermore, managers will have to retire eventually, and a lot of time and effort must be spent to replace them.

Solution

The proposed system will be simple for employees to use, and since the learning curve for this system is so low, the training time could be reduced from a few hours to just a few minutes. Rather than requiring a manager to micromanage each employee directly, employees can be given clear instructions on what needs to be done through the Employee Application. This will also augment a manager's ability to supervise employees. With this system, employee turnaround will not be an issue because the system will be simple to learn.

Section 1.5: Costs

This project will require tablets to be installed in each aisle, RFID tags for each item in the store, and RFID readers for checkout and returns.

Android tablets may be bought for under $50 each when bought in bulk. 10 tablets should suffice to be used in the Customer Assistance Terminal. 6 tablets should suffice for use in the Checkout Terminal. 1 tablet should suffice for use in the Returns Terminal. The tablets would cost approximately $850.

RFID tags can be bought for $0.05 or less when bought in bulk. Supermarkets hold around 15,000 to 60,000 different products.[3] At least 100 of each product are in stock. The example supermarket shall contain 200,000 items. The RFID tags would cost approximately $10,000; however, this price can be offset by increasing the price of products by a small amount.

Smaller RFID readers can cost around $20. These should be used to check prices, so there should be one for each Customer Assistance Terminal, which in total should be 10. Larger RFID readers can cost around $200 each. There should be one for each basket for items to be reshelved, for each bagging area, and for the return terminal. This should be 17. In total, this is approximately $3,600.

Business RFID writers would cost less than $100. Buying 10 of these would cost less than $1,000

Optionally, for the convenience of employees, store tablets or smartphones may be provided to run the application (as opposed to the employee's personal device). Assuming there are 30 employees working at all times, this could cost around $1,500.

At the beginning of integrating the system into an actual supermarket, there is a large time cost. Information about items would need to be entered into the database. Specifically, the **items** and **stock** databases would need to be created with the information of current inventory. An account would need to be made for each manager and employee. Other tables will be automatically generated based on future actions (ex. creating tasks, making sales).

After the initial integration, there will be some time cost in updating the **stock** table and tagging items with RFID tags. It will need to be updated when new stock comes in and is stored in the back room and when stock is moved from the back room to shelves. The latter task can be simplified by adding a feature where an entry for items is selected, and an amount of items can be selected to be moved from the back room. This feature would automatically decide on the location in the store based on the item's type.

Section 2: Glossary of Terms

Checkout Terminal: This is a set of a tablet, an RFID scanning bagging area, and an RFID scanning basket. The bagging area will have plastic bags for customers to use and an RFID scanner to detect all the items in the bagging area. The RFID scanning basket will be a basket with an RFID scanner that can determine the items that need to be reshelved.

Customer Application: This is an application that will allow customers to find items and check prices of items.

Customer Assistance Terminal: This is a set of a tablet, RFID scanner, and a RFID scanning basket to be located at the end of each aisle. The tablet will run a program like the Customer Application without the login option. The RFID scanning basket will be a basket with an RFID scanner that can determine the items that need to be reshelved.

Employee Application: This is an application to be installed on an employee's smart device. It will synchronize tasks between employees and alert employees of tasks.

Manager Application: This is an application to be installed on a manager's smart device. It will allow managers to monitor employees.

Returns Terminal: This is a set of a tablet and an RFID scanning basket. The tablet will contain an application for returns. The RFID scanning basket will be a basket with an RFID scanner that can determine the items that need to be reshelved.

Section 3: System Requirements

Section 3.1: Enumerated Functional Requirements

Customer Application

Identifier	Priority (5 is highest priority)	Requirement
REQ-01	5	The Customer Application will be able to look up the location and price of any item in the store.
REQ-02	3	The Customer Application will allow users to register and login to use shopping list functionality.
REQ-03	3	The Customer Application will be able to save a shopping list for registered users.
REQ-04	3	The Customer Application will be able to order a shopping list for pick up or delivery.
REQ-05	2	The Customer Application will be available to use on either in store Customer Assistance Terminals or on a user's smart device.

Customer Assistance Terminal

Identifier	Priority (5 is highest priority)	Requirement
REQ-06	4	The Customer Assistance Terminal will be able to check the price of a held item.
REQ-07	5	The Customer Assistance Terminal will be able to look up the location and price of any item in the store.
REQ-08	4	The Customer Assistance Terminal will be able to request an employee for assistance.
REQ-09	2	The Customer Assistance Terminal will be able to hold and inventory unwanted for reshelving.
REQ-10	4	The Customer Assistance Terminal will notify an employee when items are waiting to be reshelved.
REQ-11	2	The Customer Assistance Terminal will be able to open the

		Customer Application.

Checkout Application/Checkout Terminal

Identifier	Priority (5 is highest priority)	Requirement
REQ-12	5	The checkout application will be able to determine all items in the bagging area.
REQ-13	3	The checkout application will be able to lock all current items in the bagging area so they can be moved to make room for more items.
REQ-14	5	The checkout application will be able to complete the purchase transaction.
REQ-15	4	The checkout application will be able to request an employee for assistance.
REQ-16	4	The checkout application will allow an employee to log into the application to manually add or remove items.

Return Application/Return Terminal

Identifier	Priority (5 is highest priority)	Requirement
REQ-17	5	The Return Application will require an employee or manager to log into the application with their credentials before use.
REQ-18	5	The Return Application will be able to check all items in the return basket.
REQ-19	5	The Return Application will be able to refund the items to the customer who returned them.
REQ-20	1	The Return Application will check the receipt to ensure those items were bought by that customer.
REQ-21	3	The Return Application will be able to notify an employee when there are items to be put back on shelves after a return.

Employee Application

Identifier	Priority (5 is highest priority)	Requirement
REQ-22	5	The Employee Application will require all employees to log into the application with their credentials before use.
REQ-23	3	The Employee Application will log employee hours by allowing employees to clock in and clock out of work.
REQ-24	3	The Employee Application will monitor item stock on shelves.
REQ-25	5	The Employee Application will maintain a list of tasks that have been done and still need to be done.
REQ-26	3	The Employee Application will create a task and notify employees when shelves need to be restocked.
REQ-27	5	The Employee Application will create a task and notify employees when items need to be reshelved from the Customer Assistance Terminals or the Return Terminals.
REQ-28	1	The Employee Application will create a task and notify employees when items are about to expire and need to be removed from shelves.
REQ-29	5	The Employee Application will allow employees to view tasks that still need to be done.
REQ-30	4	The Employee Application will allow employees to claim a task to let other employees know they are doing that task.
REQ-31	3	The Employee Application will allow employees to mark a task as complete.
REQ-32	2	The Employee Application will allow employees to call a manager for assistance.
REQ-33	2	The Employee Application will play a clear alert tone when a new task is available.

Manager Application

Identifier	Priority (5 is highest priority)	Requirement

REQ-34	5	The Manager Application will require all managers to log into the application with their credentials before use.
REQ-35	3	The Manager Application will monitor stock in inventory.
REQ-36	4	The Manager Application will allow managers to view and correct employee work hours.
REQ-37	5	The Manager Application will allow managers to view the history of tasks from the employee application and their completion.
REQ-38	3	The Manager Application will alert managers if there is low stock in the inventory.
REQ-39	2	The Manager Application will alert managers when an employee needs assistance.
REQ-40	2	The Manager Application will allow managers to mark that they are going to assist an employee.
REQ-41	4	The Manager Application will calculate shop statistics such as sales and returns figures.
REQ-42	5	The Manager Application will allow managers to manually add tasks to the employee task list.
REQ-43	5	The Manager Application will calculate an employee’s salary.
REQ-44	4	The Manager Application will allow registration of new employees or managers.

Section 3.2: Enumerated Nonfunctional Requirements

Customer Application/Customer Assistance Terminal

Identifier	Priority (5 is highest priority)	Requirement
REQ-45	4	The Customer Assistance Terminal should be clearly visible and easy to spot in the store.
REQ-46	4	The Customer Assistance Terminal should allow simple price checking by only requiring the item be held near the RF sensor.
REQ-47	4	The Customer Assistance Terminal should accept unwanted items by simply placing them into the reshelving basket.

Checkout Application/Checkout Terminal

Identifier	Priority (5 is highest priority)	Requirement
REQ-48	4	The Checkout Terminal and application should provide clear and easy instructions on how to checkout.

Return Application/Return Terminal

Identifier	Priority (5 is highest priority)	Requirement
REQ-48	4	The Return Terminal and application should clearly display items being returned for easy verification with the employee processing the return.

Employee Application

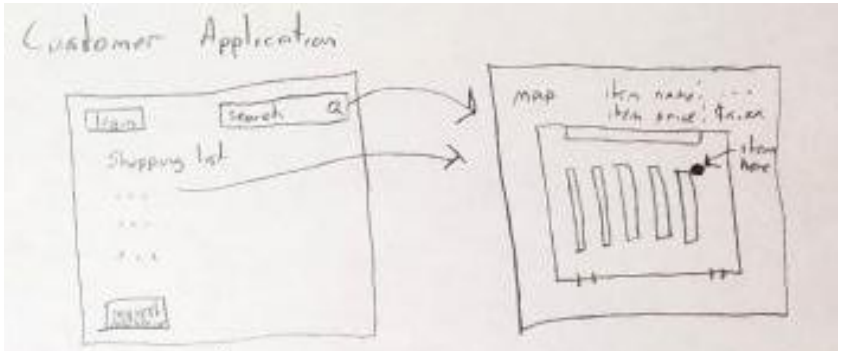
Identifier	Priority (5 is highest priority)	Requirement
REQ-49	5	The Employee Application should make it clear from the home screen what new tasks need to be done.
REQ-50	3	The Employee Application should be streamlined as to not busy employees with using the application.

Manager Application

Identifier	Priority (5 is highest priority)	Requirement
REQ-51	5	The Manager Application should be able to easily view working tasks and tasks that still need to be done to ensure tasks are being completed in a timely manner.

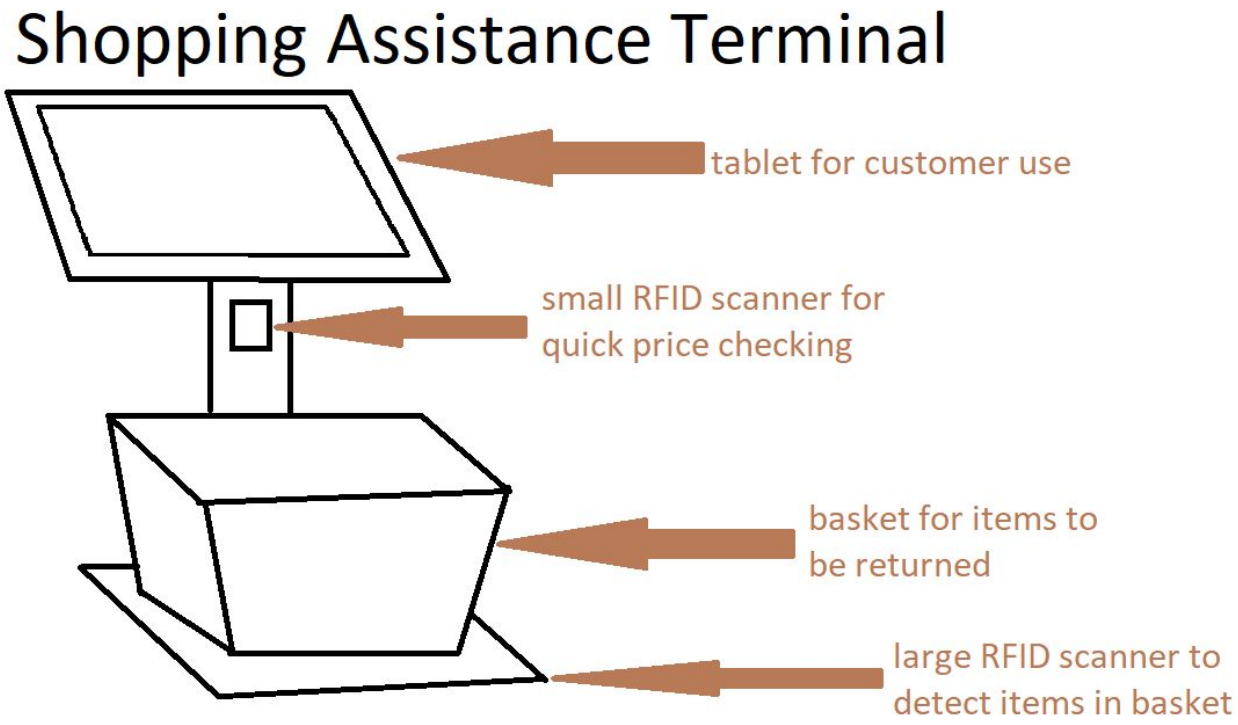
Section 3.3: User Interface Requirements

Customer Application



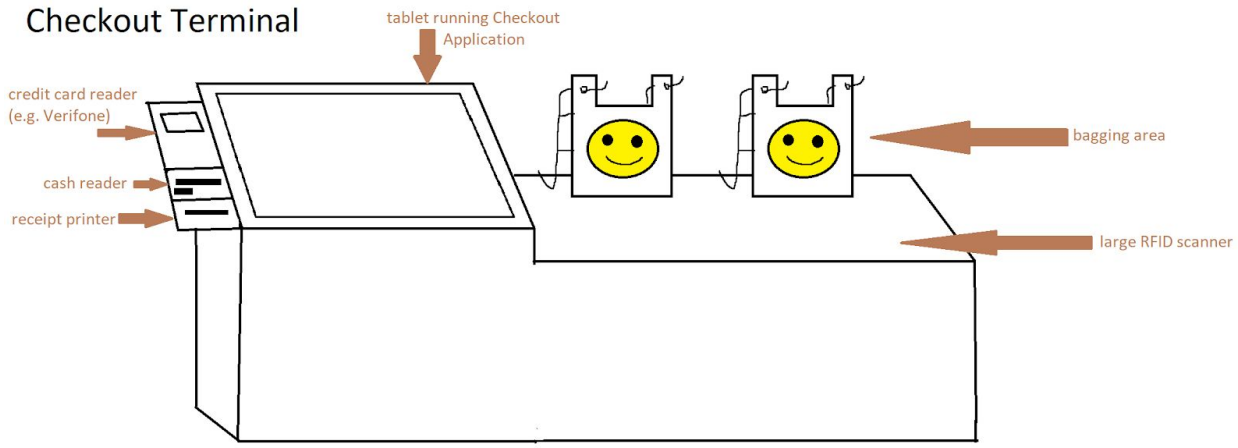
Identifier	Priority (5 is highest priority)	Requirement
REQ-52	5	The Customer Application should have a search box for items on the home page.
REQ-53	5	The Customer Application should display the location and price of an item if it is searched for.
REQ-54	4	The Customer Application should display the location and price of an item selected from a customer's shopping list.

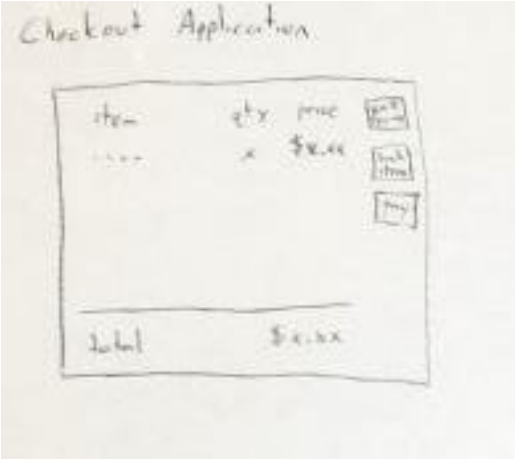
Customer Assistance Terminal



Identifier	Priority (5 is highest priority)	Requirement
REQ-55	5	The Customer Assistance Terminal should display the price and name of an item if it is scanned for a price check.
REQ-56	4	The Customer Assistance Terminal should have a clearly marked basket for unwanted items.

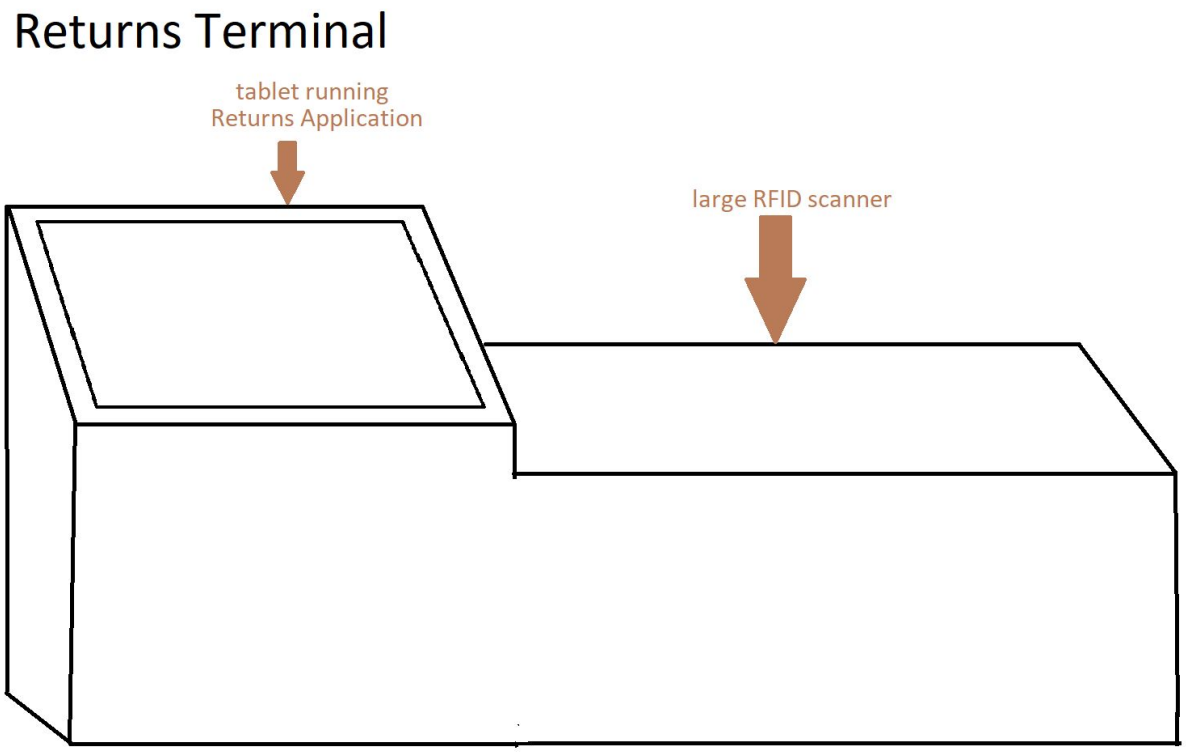
Checkout Application/Checkout Terminal

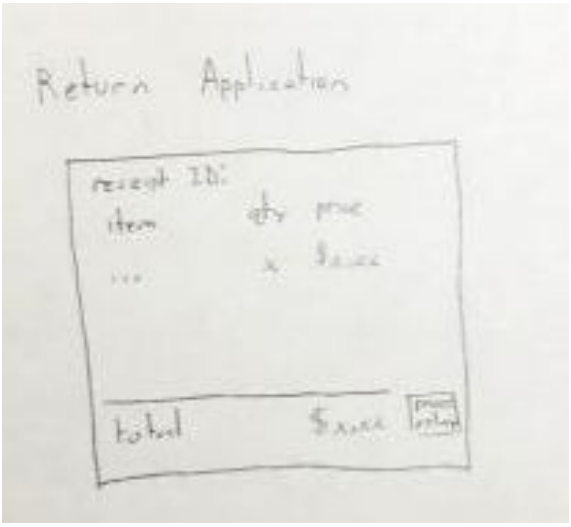




Identifier	Priority (5 is highest priority)	Requirement
REQ-57	4	The Checkout Terminal should have clear buttons for “checkout”, “lock cart” and “call for assistance”.
REQ-58	5	The Checkout Terminal should display the name, price, and amount of all products that have been scanned.
REQ-59	3	The Checkout Terminal should have a clearly marked basket for unwanted items.

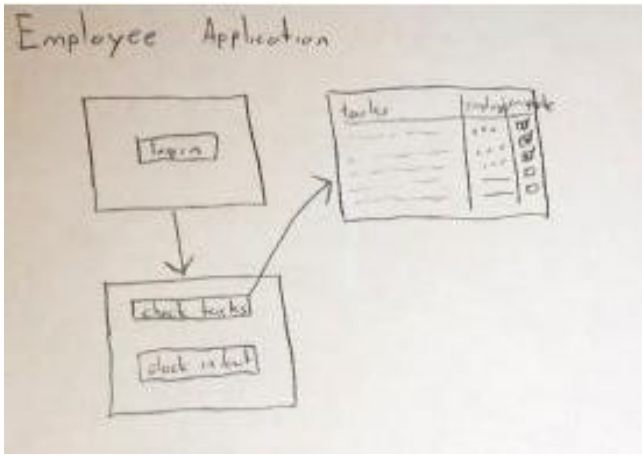
Return Application/Return Terminal





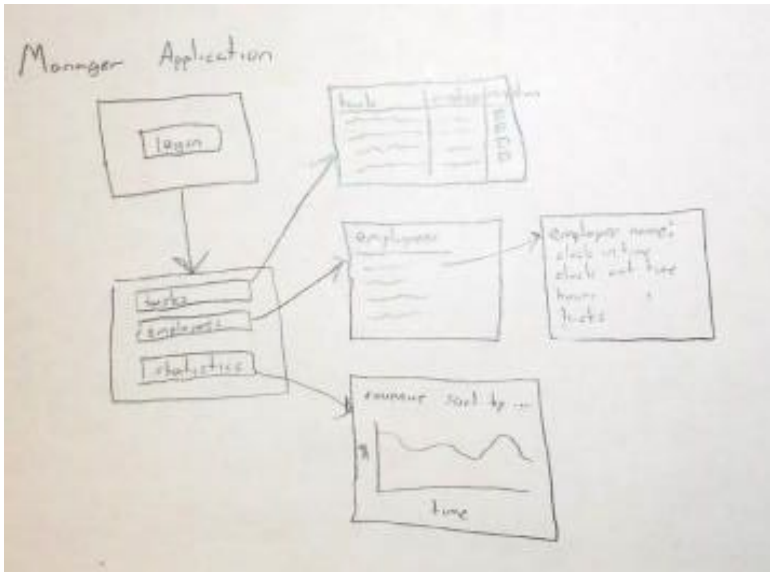
Identifier	Priority (5 is highest priority)	Requirement
REQ-60	3	The Return Terminal should display all items on a scanned receipt and allow the items to be selected.

Employee Application



Identifier	Priority (5 is highest priority)	Requirement
REQ-61	5	The Employee Application will display all unclaimed, uncompleted tasks and will have buttons to allow an employee to claim that task.
REQ-62	3	The Employee Application will display an employee's claimed task(s) for that employee.
REQ-63	4	The Employee Application will have buttons to allow an employee to mark their claimed task(s) as complete.

Manager Application



Identifier	Priority (5 is highest priority)	Requirement
REQ-64	5	The Manager Application will have buttons so that managers can choose to display tasks, employees, or statistics.
REQ-65	5	The Manager Application will display all tasks, their state of completion, and the employee who claimed the task (if applicable) in the task menu.
REQ-66	3	The Manager Application will display employee names in the employee menu.
REQ-67	3	The Manager Application will display an employee's clock in time, clock out time (if applicable), hours for the day (if applicable), claimed and completed tasks when an employee is selected in the employee menu.
REQ-68	3	The Manager Application will display sorted revenue statistics in the statistics menu.

Section 4: Functional Requirements Specification

Section 4.1: Stakeholders

The stakeholders for the store automation system are supermarket owners, customers, and employees. Supermarket owners will use this system to automate their stores and expand their business. Supermarket customers will use tablets placed in the store or use a phone application. The supermarket employees will have to learn to use the Employee Application and depending on their position, the Manager Application as well in order to best assist customers.

Section 4.2: Actors and Goals

- **Customer Application (Initiating) -** This application will assist customers in finding an item and checking its price. Also, the application can call for an employee's help and will send them an alert for reshelving items. The application will be able to save a customer's shopping list.
- **Checkout Application (Initiating) -** This application will be able to check all the items that a customer wants to buy and will complete the transaction. It will be able to call for an employee's help. Also, it allows an employee to manually add or remove items.
- **Return Application (Initiating) -** This application will be able process all returned items. It will check all items and check if it satisfies the return policy set by the store. After that it will give the customer a refund on the item. Also, it will notify an employee to put the returned item back.
- **Employee Application (Initiating) -** This application will monitor an employee's hours and pay. It will show what tasks employees need to do and what still remains.
- **Manger Application (Initiating) -** This application will oversee everything, especially the checking of stock. Furthermore, it will help calculate total revenue after each week and total cost to operate the store. It will monitor employee's work schedules and will be able to make any changes. It will notify the manager if an employee needs extra help
- **Customers (Participating)** - They will buy products from the store. They will use the Customer Application to check prices and find an item that they want. After they get everything they wanted, they will use the checkout application to finish their transaction. Also, they will be able to return an item at a Return Terminal.
- **Employees (Participating)** - The employees will assist customers if they need help. They will reshelve items and put the returned items on shelves. They will know how much money they are making.
- **Manger (Participating)** - The manager will oversee everything that goes in the store. The manager will know how much inventory is in the store and will be able to order more products if any are out of stock. Also, the manager will assign employees tasks to complete.

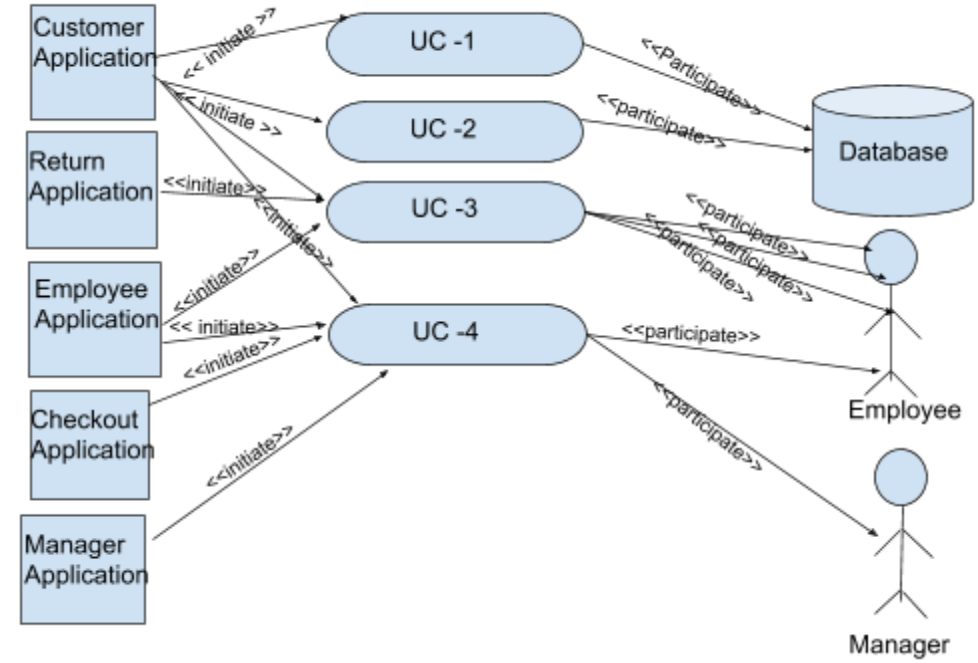
Section 4.3: Use Cases

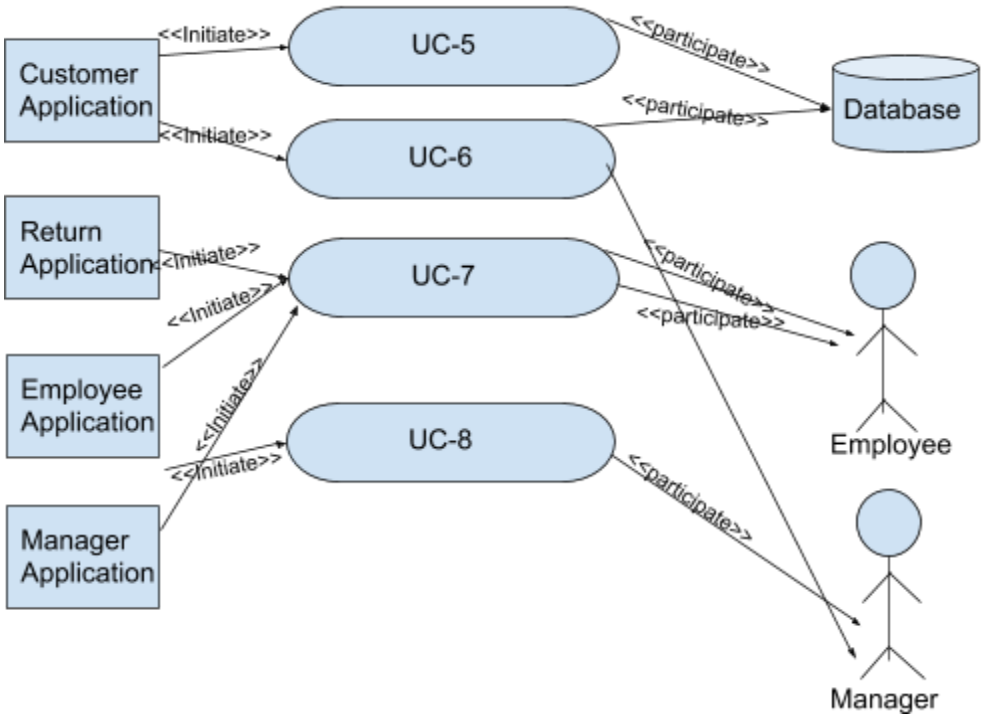
Section 4.3.1: Casual Description

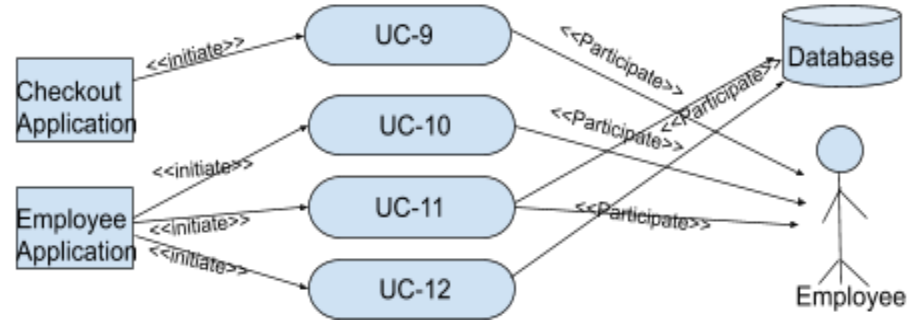
Use Case	Description	REQ
PriceChecker (UC-1)	To check and display the price of an item after scanning an item's RF tag with a sensor	REQ-1 RE-42 REQ-52
ItemLocator (UC-2)	To search for and display the location of any item from the customer assistant terminal.	REQ-2 REQ-50 REQ-51
ItemReshelve (UC-3)	To alert employees when an item needs to be reshelved and shelves restocked	REQ-8 REQ-18 REQ-23
AssistantCaller (UC-4)	To alert employees or managers when a customer is in need of assistance	REQ-3 REQ-12 REQ-29 REQ-36
ShoppingCart (UC-5)	To store a list of items the customer wishes to purchase in a virtual shopping cart on the customer application.	REQ-4
OrderItems (UC-6)	To place an order of items for pick-up or delivery through the customer application	REQ-5 REQ-53
LogIn (UC-7)	To allow access to employee and manager applications only when an employee or manager enters their credentials	REQ-14 REQ-19 REQ-31
Checkout (UC-8)	To facilitate smooth purchase transactions through the checkout application	REQ-11 REQ-44 REQ-55
EmployeeOverride (UC-9)	To allow employees the ability to manually add or remove items from the bagging area	REQ-13
CreateTask (UC-10)	To add task to the task board of required jobs such as reshelving items or discarding expired items	REQ-24 REQ-25

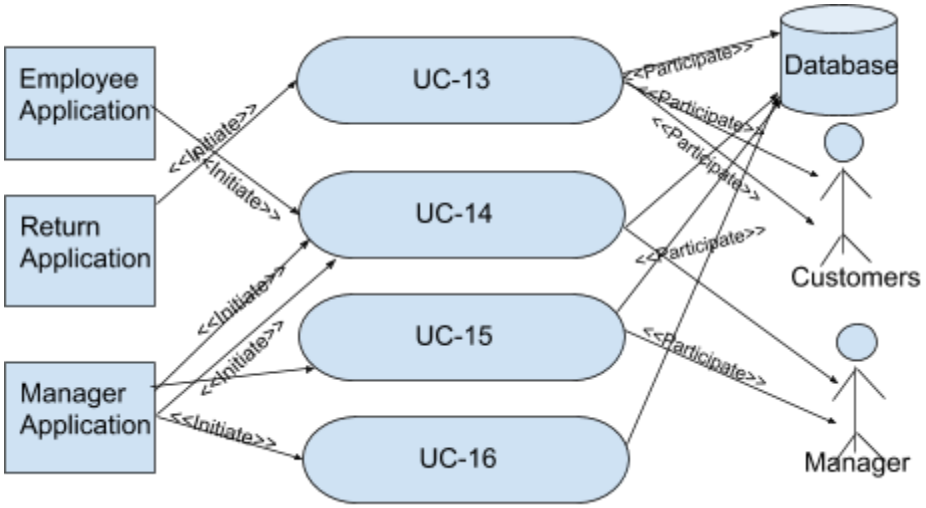
TaskBoard (UC-11)	To provide a simple interface for employees to view, claim, or mark complete tasks.	REQ-22 REQ-26 REQ-27 REQ-28 REQ-58 REQ-59 REQ-60
TaskAlert (UC-12)	To send an audible alert to employees when new tasks are available and uncompleted	REQ-30 REQ-46
ReturnItem (UC-13)	To facilitate an efficient return of items process, which offers transparency of the process to the user and mechanisms to speed up multiple returns	REQ-16 REQ-17 REQ-45 REQ-57
InventoryCheck (UC-14)	To display the inventory of stock in the store and alert employees if inventory is low	REQ-21 REQ-32 REQ-35
MonitorEmployee (UC-15)	To keep track of employees' schedules and calculate salaries based on their hours	REQ-33 REQ-40
StoreStatistics (UC-16)	To calculate the total revenue and total operation cost for a given day	REQ-38 REQ-65
UnwantedItem (UC-17)	To accept items that a customer does not want until the items are reshelved	REQ-7 REQ-43 REQ-56
ItemCheck (UC-18)	To identify items obtained by the customers.	REQ-9 REQ-15
ManagerBusy (UC-19)	To mark that the manager is helping someone in order to inform employees that the manager is busy.	REQ-37

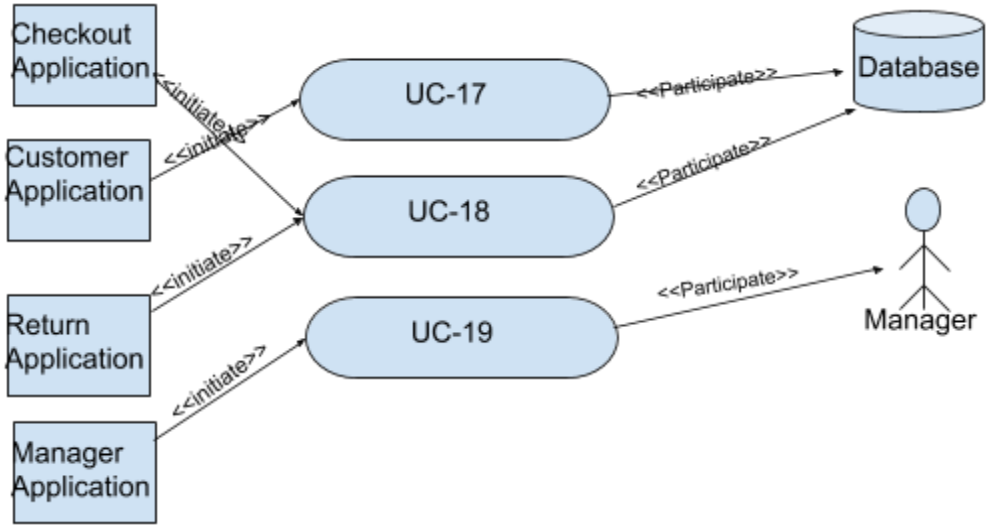
Section 4.3.2: Use Case Diagrams






Checkout Application
Employee Application
UC-9
UC-10
UC-11
UC-12
<<initiate>>
<<Participate>>
Database
Employee


Employee Application
Return Application
Manager Application
UC-13
UC-14
UC-15
UC-16
<<Initiate>>
<<Participate>>
Database
Customers
Manager


Checkout Application
Customer Application
Return Application
Manager Application
UC-17
UC-18
UC-19
<<initiate>>
<<Participate>>
Database
Manager

Section 4.3.3: Traceability Matrix

REQ-#	PW	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11	UC-12	UC-13	UC-14	UC-15	UC-16	UC-17	UC-18	UC-19
REQ-1	4	X																		
REQ-2	5		X																	
REQ-3	4				X															
REQ-4	3					X														
REQ-5	3						X													
REQ-6	2																			
REQ-7	2																	X		
REQ-8	4			X																
REQ-9	5																		X	
REQ-10	3																			
REQ-11	5								X											
REQ-12	4				X															
REQ-13	3									X										
REQ-14	5							X												
REQ-15	5																		X	
REQ-16	5													X						
REQ-17	1													X						
REQ-18	3			X																
REQ-19	5							X												
REQ-20	3																			
REQ-21	3														X					
REQ-22	5											X								

REQ-23	3			X															
REQ-24	5										X								
REQ-25	1										X								
REQ-26	5											X							
REQ-27	4											X							
REQ-28	3											X							
REQ-29	2				X														
REQ-30	2												X						
REQ-31	5							X											
REQ-32	3														X				
REQ-33	4															X			
REQ-34	5																		
REQ-35	3														X				
REQ-36	2				X														
REQ-37	2																		X
REQ-38	4																X		
REQ-39	5																		
REQ-40	5															X			
REQ-41	4																		
REQ-42	4	X																	
REQ-43	4																	X	
REQ-44	4								X										
REQ-45	4													X					
REQ-46	5												X						
REQ-47	3																		

REQ-48	5																			
REQ-49	4																			
REQ-50	5		X																	
REQ-51	5		X																	
REQ-52	5	X																		
REQ-53	4						X													
REQ-54	4																			
REQ-55	5								X											
REQ-56	3																	X		
REQ-57	3												X							
REQ-58	5											X								
REQ-59	3											X								
REQ-60	4											X								
REQ-61	5																			
REQ-62	5																			
REQ-63	3																			
REQ-64	3																			
REQ-65	3																X			
Max PW		5	5	4	4	3	4	5	5	3	5	5	5	5	3	5	4	4	5	2
Total PW		**13**	**15**	**10**	**12**	**3**	**7**	**15**	**14**	**3**	**6**	**6**	**7**	**13**	**9**	**9**	**7**	**9**	**10**	**2**

Section 4.3.4: Fully-Dressed Descriptions

UC-02	**ItemLocator**
Related Requirements	2, 50, 51
Initiating Actor	Customer
Actor's Goal	To receive information about an item's location in the store
Participating Actors	Customer
Preconditions	Action performed through the customer assistance terminals
Postconditions	User receives map and aisle data of requested item
Flow of Events (Typical Scenario Using Search Box)	
→ **1.**	User enters the search box
← **2.**	While user searches, best-match items are displayed in a dropdown. Only items on stock are available. If an item is out of stock, the item name will appear greyed out with a symbol besides it to symbolize its status as out of stock.
→ **3.**	User selects best match from dropdown menu
↬ **4.**	System searches for item data
← **5.**	A very simple map of the store is shown with the aisle of the item highlighted. Aisle name and number is displayed along with the map.

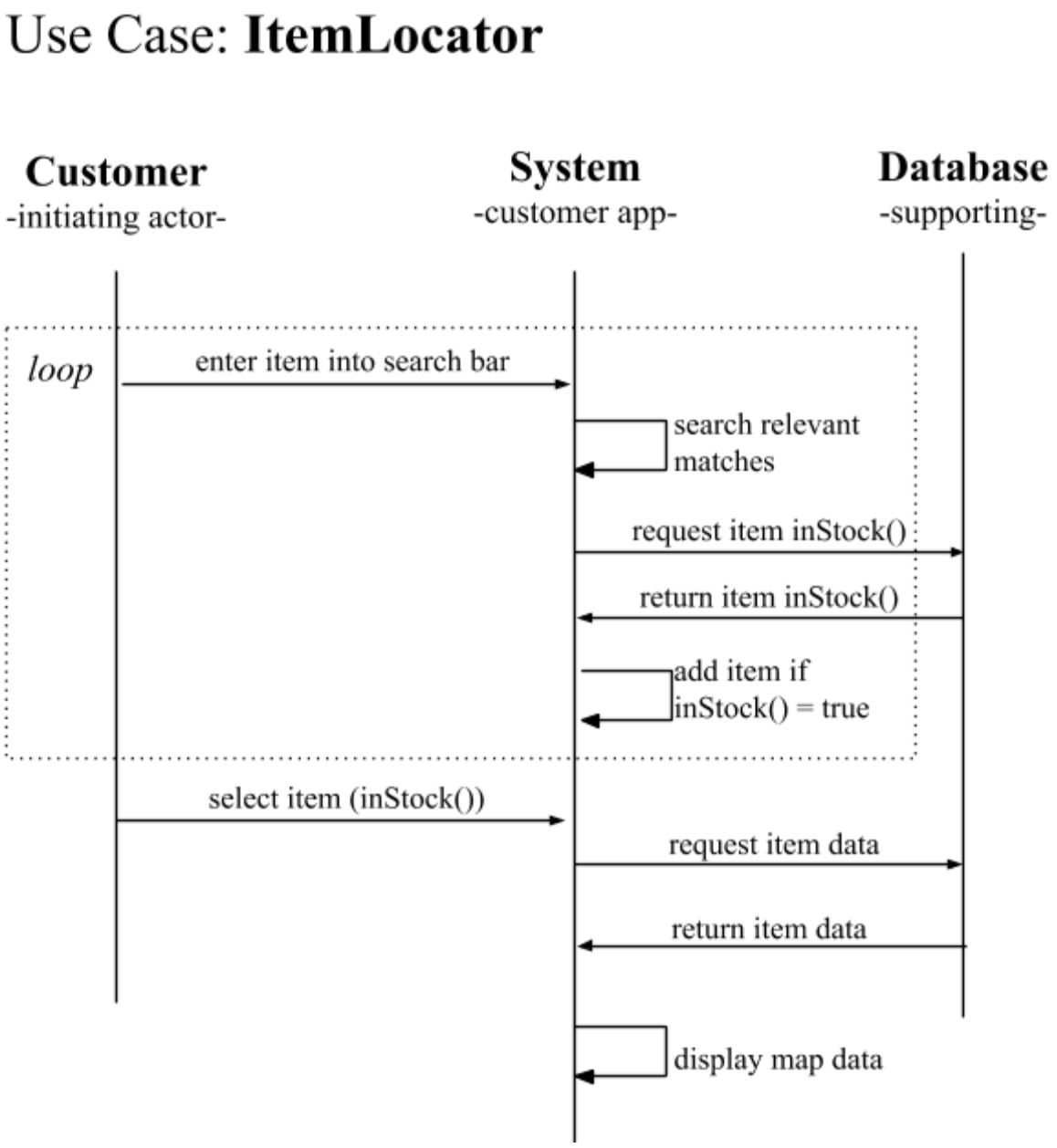
UC-07	**LogIn**
Related Requirements	14, 19, 31
Initiating Actor	Employee/Manager
Actor's Goal	Allow access to employee and manager applications only when an employee or manager enters their credentials
Participating Actors	Employee, Manager
Preconditions	Employee and Manager applications features are unusable
Postconditions	Employee and Manager applications features are usable
Flow of Events (First time employee login)	
→ **1.**	User opens application
← **2.**	Application presents login screen with suboption for users(employees) to create a new account
→ **3.**	User selects "create account"
← **4.**	App requests user to enter credentials for account creation
→ **5.**	User inputs credentials and sends a request for account to be approved by manager
← **6.**	Manager receives request from employee to create an account
→ **7.**	Manager accepts request from employee
← **8.**	Employee app grants employee access to employee features.
Flow of Events (Login with pre-existing credentials)	
→ **1.**	User opens application
← **2.**	Application presents login screen with an input text field
→ **3.**	User inputs credentials
↬ **4.**	System verifies credentials
← **5.**	On verification completion, user is given access to account specific features

UC-08	**Checkout**
Related Requirements	11, 44, 55
Initiating Actor	Customer
Actor's Goal	To efficiently checkout items for purchase
Participating Actors	Employee
Preconditions	Customer arrives at a checkout terminal to begin checkout
Postconditions	Customer's items have been purchased and the necessary inventory data is updated.
Flow of Events (Successful Checkout Scenario)	
→ 1.	Customer presses button to begin checkout
← 2.	Terminal prompts customer to bag each item
↬ 3.	During bagging, RFID reader scans items in bagging area
← 4.	For each item in bagging area, the terminal displays the item's name, price, and quantity (if multiple of the same item is scanned), and puts the item in a list of total items for purchase.
→ 5.	Customer presses a button indicating the checkout is finished
← 6.	Terminal prompts user to select payment method (cash or credit)
↬ 7.	Once payment is processed, inventory data is updated and checkout is complete
Flow of Events (Alternate Scenarios)	
→ 1.	Customer presses a button to get assistance from an employee
↬ 2.	System sends task alert to alert an employee that a customer at terminal A needs assistance

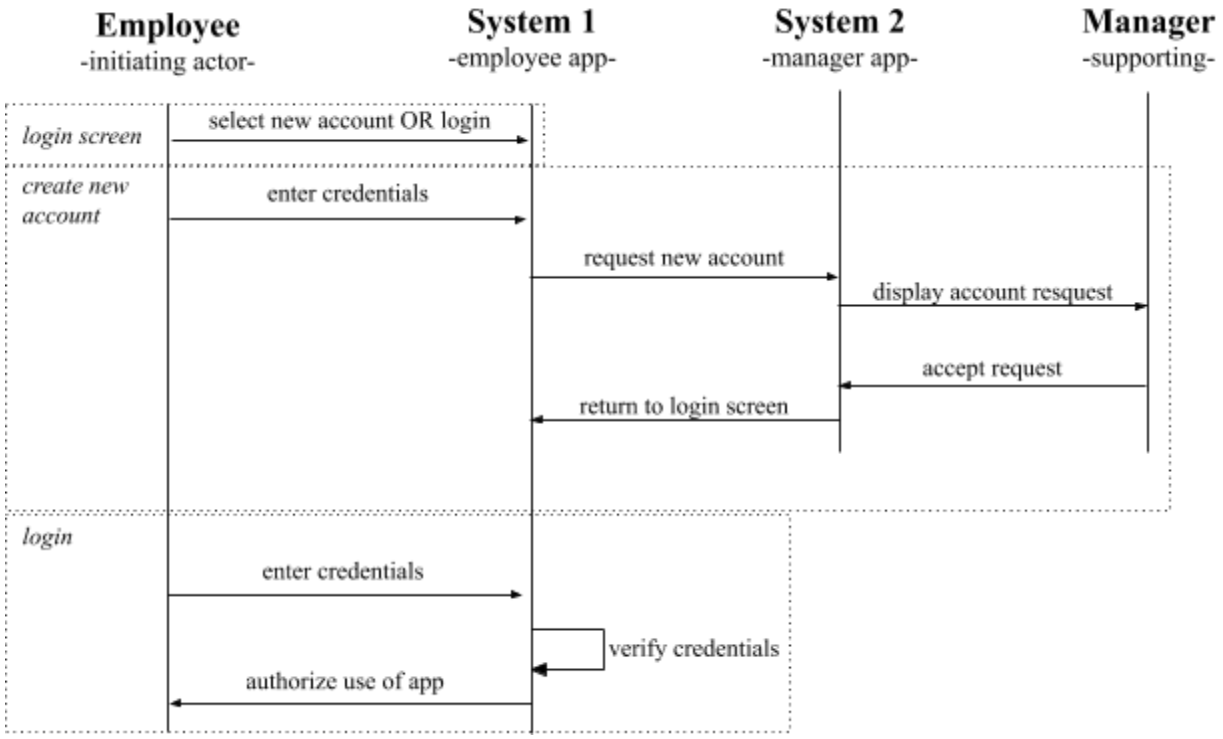
UC-14	**ReturnItem**
Related Requirements	16, 17, 45, 57
Initiating Actor	Customer
Actor's Goal	To return purchased items with a refund
Participating Actors	Employee, Customer, Return Terminal (System)
Preconditions	Customer must provide proof of purchase through receipt
Postconditions	Item having met return policy is returned to shelves
Flow of Events (Successful Return Scenario)	
→ 1.	Customer scans/enters all items to be returned
→ 2.	System calls employee for verification
← 3.	Employee logs in
← 4.	Employee verifies items to be returned
↬ 5.	System accepts verification and proceeds to ask customer for a refundable source
→6.	Customer selects card of purchase or enters a card
↬ 7.	System refunds item costs to selected method
Flow of Events (Unsuccessful Return Extension)	
← 4a.	If Employee deems item to be invalid, then indicates that on the system and tells the user

UC-01	**PriceChecker**
Related Requirements	1, 42, 52
Initiating Actor	Customer
Actor's Goal	To receive information about an item's price
Participating Actors	
Preconditions	Action performed through a customer assistance terminal
Postconditions	User has obtained the price of an item
Flow of Events (Success Scenario)	
→ **1.**	User scans item in RFID scanner
↬ **2.**	System searches database for item
← **3.**	System returns the price of item

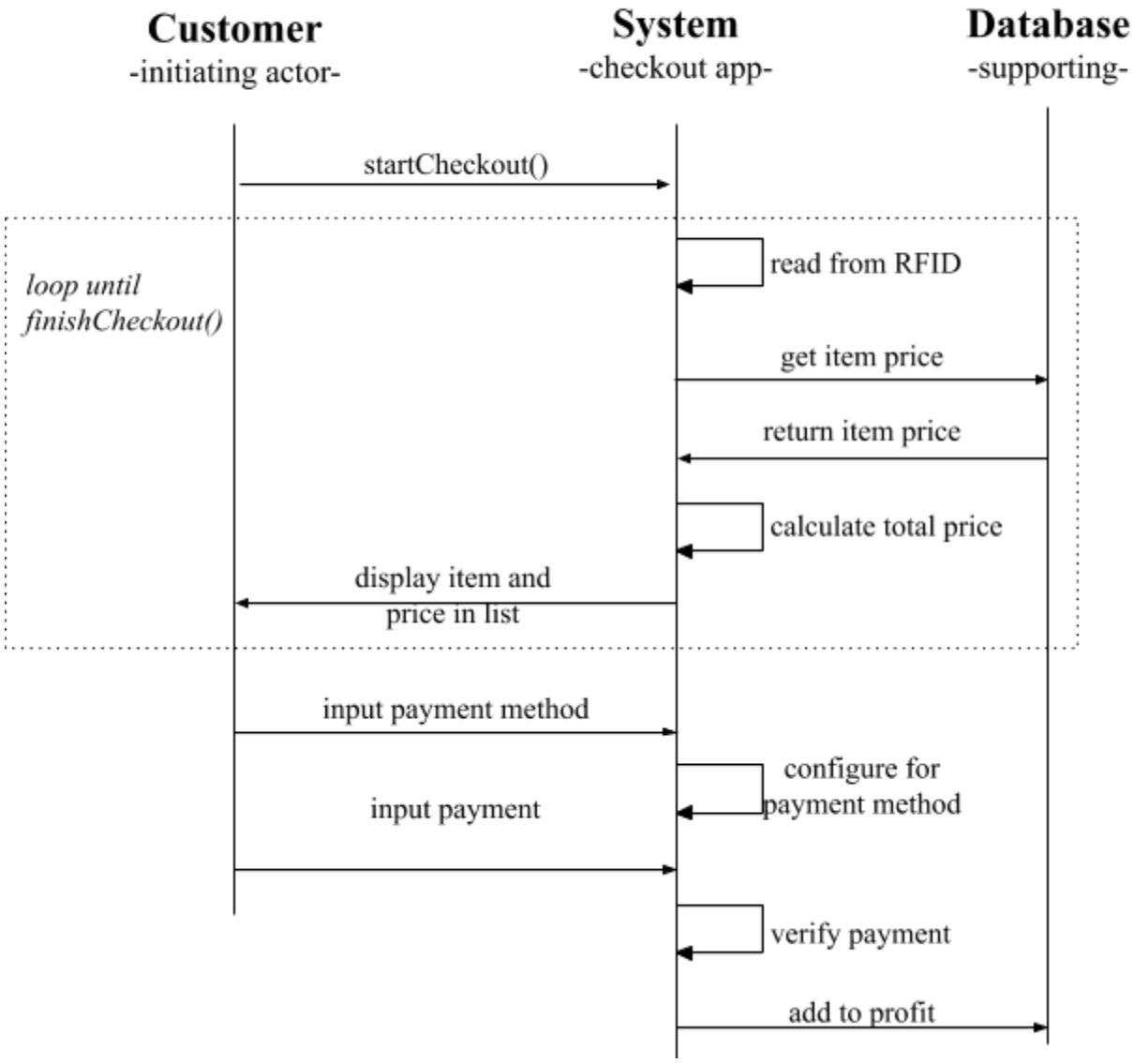
Section 4.4: System Sequence Diagrams



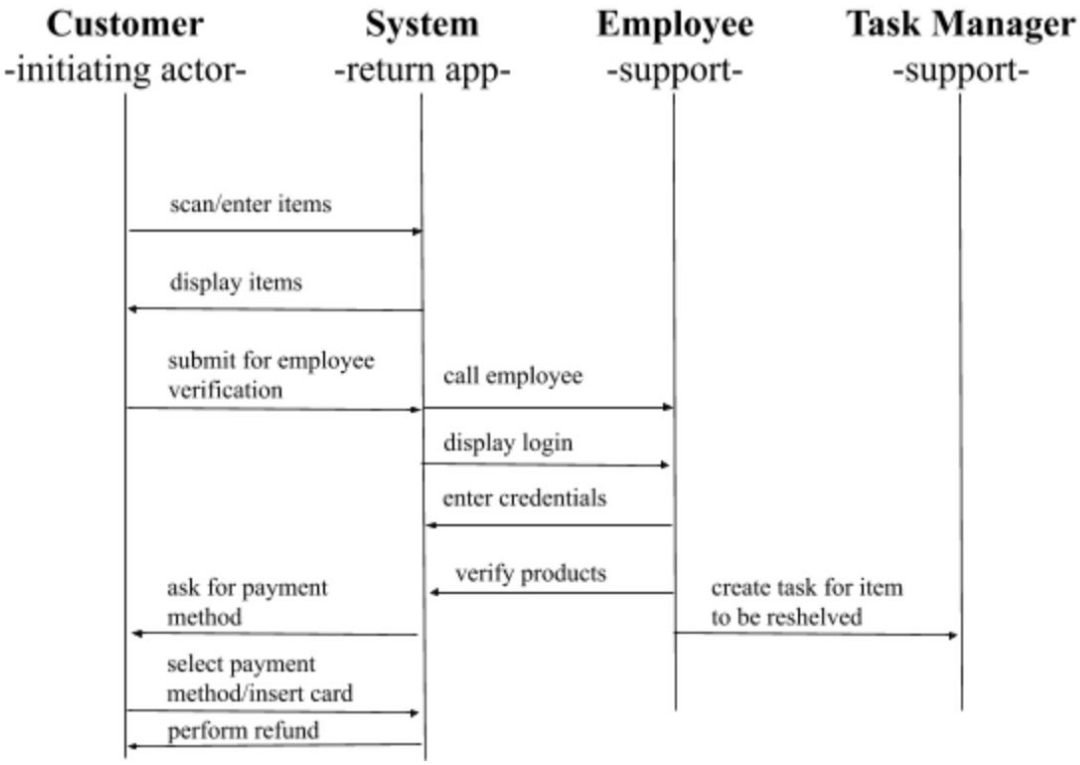
Use Case: **LogIn**



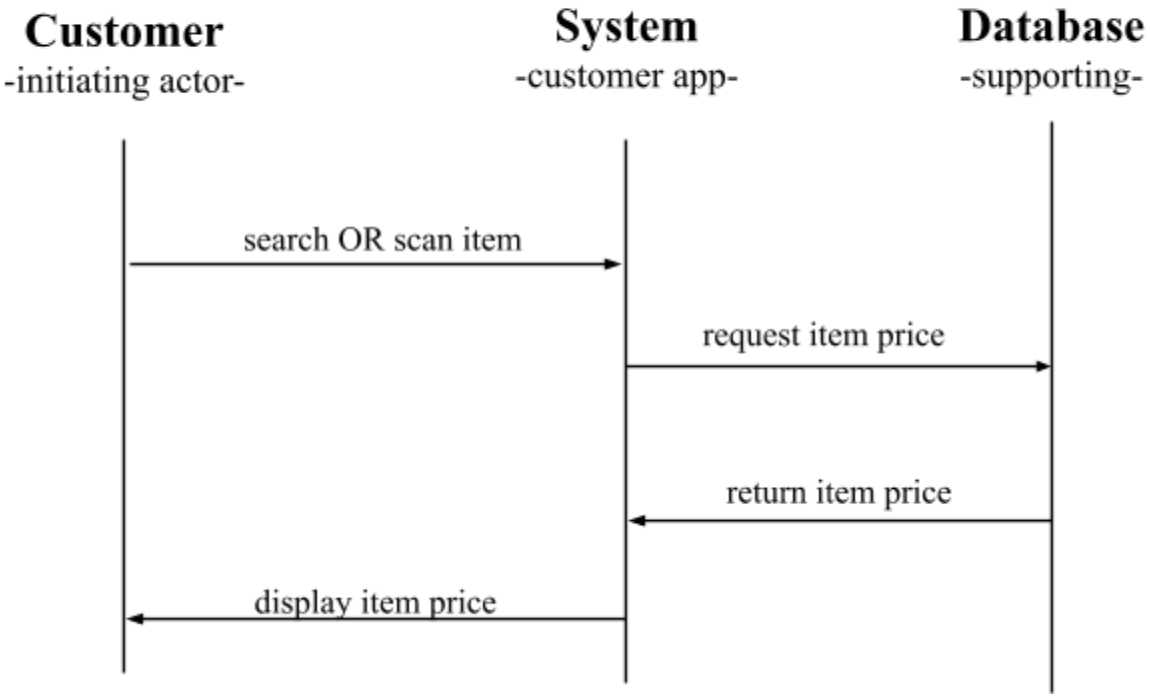
Use Case: **Checkout**



Use Case: **ReturnItem**



Use Case: **PriceChecker**



Section 5: Effort Estimation using Use Case Points

	UUCW	UAW				Weights	
UC-1	10	1		T1	4	2	8
UC-2	5	3		T2	5	1	5
UC-3	10	2		T3	4	1	4
UC-4	5	3		T4	2	1	2
UC-5	10	3		T5	4	1	4
UC-6	5	3		T6	1	0.5	0.5
UC-7	10	3		T7	5	0.5	2.5
UC-8	10	3		T8	2	2	4
UC-9	5	3		T9	3	1	3
UC-10	5	2		T10	4	1	4
UC-11	10	3		T11	5	1	5
UC-12	5	2		T12	1	1	1
UC-13	10	3		T13	1	1	1
UC-14	10	2					
UC-15	5	1					
UC-16	10	1					
UC-17	5	1					
UC-18	10	2					
UC-19	10	1					
Total	150	42		TCF	1.04		
UCP	199.68						

This spreadsheet was made in Google Sheets to allow for flexibility and organization in solving for the UCP. The UCP is calculated by UCP = UUCP *TCF*ECF, though for the purposes of this project we are taking the ECF as 1, and UUCP = UUCW + UAW which are 150 and 42 respectively. Then using the UCP calculated and multiplying by our Productivity Factor we would get an effort Estimation of 5,591 hours.

Section 6: Domain Analysis

Section 6.1: Domain Model

Section 6.1.1: Concept Definitions

Responsibility Description	Type	Concept
Coordinate actions of all concepts associated and interacting with a system (app/terminal)	D	Controller (customer, checkout, return, employee, manager)
Holds all information about an item	K	Item
Stores price info about an item (Database)	D	PriceHolder
Stores info about amount of each item on shelves and in storage (Database)	D	StockManager
Stores info about item's aisle number, map data (Database)	D	ItemLocator
Creates and delegates tasks (including alerts) for employee and manager	D	TaskManager
A job that needs to be completed by an employee or manager	K	Task
Holds list of all Employee objects	D	EmployeeDatabase
Database of the company's financial data (profit, transactions, etc.)	D	FinancialDatabase
Contains a list of items placed in return bins to be restocked	D	Bin
Contains a map of the store and functions that highlight different areas given an aisle and section number	D	MapMaker
Sends information to user through a user interface	D	Terminal

Section 6.1.2: Association Definitions

Concept Pair	Association Description	Association
Controller PriceHolder	Controller requests the price of an item that was searched for or scanned	requests price
Controller (manager) StockManager	Controller requests storage information about an item that was searched for.	requests stock
Controller (checkout/return) StockManager	Controller updates storage information about an item that was purchased or returned.	updates stock
Controller ItemLocator	Controller requests location information about an item that was searched for or scanned. This info includes aisle number and map data for that item.	requests location
Controller MapMaker	Controller requests a map image with the aisle and section number highlighted for a specific item.	requests mapimage
Controller *(manager)* EmployeeDatabase	Controller requests employee hours	requests hours
Controller *(employee)* EmployeeDatabase	Controller (employee app) sends employee hours to database	update hours
Controller *(manager)* FinancialDatabase	Controller requests financial data	requests finances
Controller *(checkout/return)* FinancialDatabase	Controller sends profit	send profit
Controller TaskManager	Controller requests employee task data	views tasks
Controller TaskManager	Controller sends Task to task manager	creates task
Bin TaskManager	Bin sends Task to task manager	creates task

TaskManager Controller	Task Manager sends Task to controller (employee system)	displays tasks
Controller Terminal	Controller creates and sends to user a task alert of a new task upon receiving new task	sends alert
Controller Terminal	Controller sends info to terminal to display to user	displays UI

Section 6.1.3: Attribute Definitions

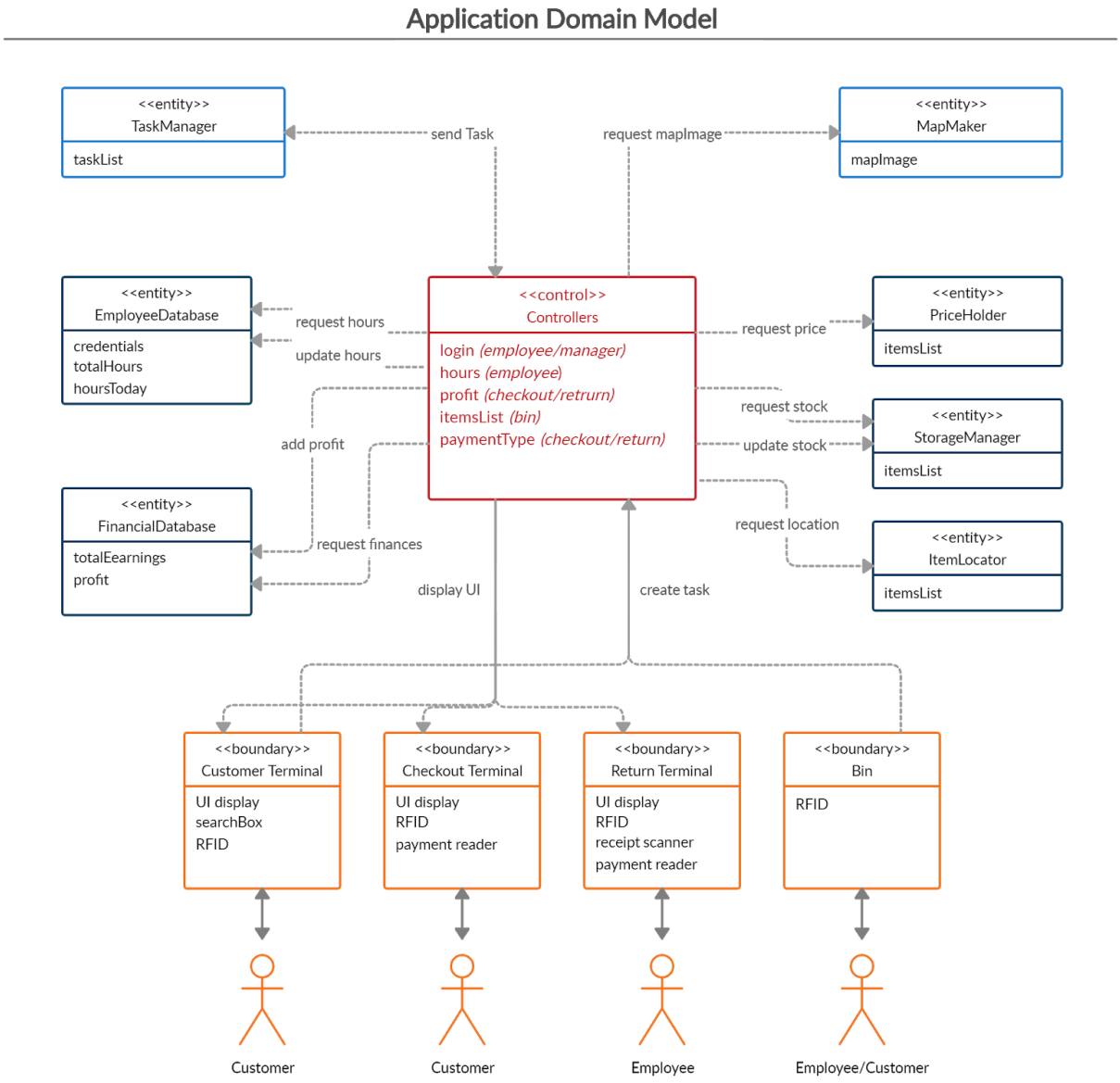
Concept	**Attributes**	**Attribute Description**
Item	name	String name of item
	price	Price of an Item
	asile	The aisle an Item is shelved in
	section	Position of Item in an aisle
	numInStock	Number of Items on shelves
	numInStorage	Number of Items in backroom
	numInBin	Number of Items not on shelves due to being returned or put in Bin
	binNumber	An identifier of which bin an Item has been placed in
	RFID	Unique identifier for Item
PriceHolder	Item list	List of items whose prices can be accessed
StockManager	Item list	List of items whose stock information can be accessed
ItemLocator	Item list	List of items whose location data can be accessed
Controller	login	Check if user is logged in to system
	hours	Employee system counts hours for each employee

	profit	Checkout and Return systems sends profit (or loss) to financial database
Bin	Item list	List of Items in all physical bins
TaskManager	TaskList	A list of all Tasks received to display
Task	details	String explaining task
	complete	Boolean to check whether or not task is complete
	employee	Identifier of which employee is working on task
EmployeeDatabase	credentials	Data needed for an employee to sign in to account
	totalHours	Total hours worked by an employee
	hoursToday	Total hours worked today
FinancialDatabase	total earnings	total earnings of company
	today's profit	Profit for a single day
MapMaker	mapimage	An image of the store generated using location data from an Item
SearchRequest	name	name of an item

Section 6.1.4: Traceability Matrix

		DOMAIN CONCEPTS							
UC-#	**PW**	Price Holder	Stock Manager	Item Locator	Map Maker	Employee Database	Financial Database	TaskManager	Terminal
UC-1	13	X	X	X			X		X
UC-2	15				X				
UC-3	10		X	X		X			
UC-4	12	X	X				X	X	
UC-5	3	X	X				X		X
UC-6	7	X	X				X		X
UC-7	15					X			
UC-8	14	X	X				X		X
UC-9	3	X	X			X	X		X
UC-10	6		X	X		X		X	X
UC-11	6					X		X	X
UC-12	7					X		X	
UC-13	13	X							
UC-14	9		X	X		X		X	
UC-15	9					X	X	X	
UC-16	7	X	X				X		
UC-17	9	X	X				X	X	
UC-18	10		X	X			X		
UC-19	2					X		X	

Section 6.1.5: Application Domain Model



Section 6.2: System Operation Contracts

Operation	**Locate Item**
Preconditions	- Item name searched in searchBox - Item stock > 0
Postconditions	- Item's location data is used by MapMaker to generate map image

Operation	**Log In**
Preconditions	- User is currently not logged into system - User has already created an account
Postconditions	- User is granted access to role specific application

Operation	**Checkout**
Preconditions	- A checkout is not currently in progress - Customer's items have valid RFID tag
Postconditions	- Customer's items are purchased (transaction successful) - Inventory databases updated - Unwanted items placed in return bins

Operation	**Return Item**
Preconditions	- Customer possess proof of purchase (receipt) - Item is compliant with return policy
Postconditions	- Items, in accordance with return policy, are placed in return bins - Customer receives necessary refund

Operation	**Check Price**
Preconditions	- Item scanned has valid RFID tag - Item is in database
Postconditions	- User obtains the price of an item

Section 7: Interaction Diagrams

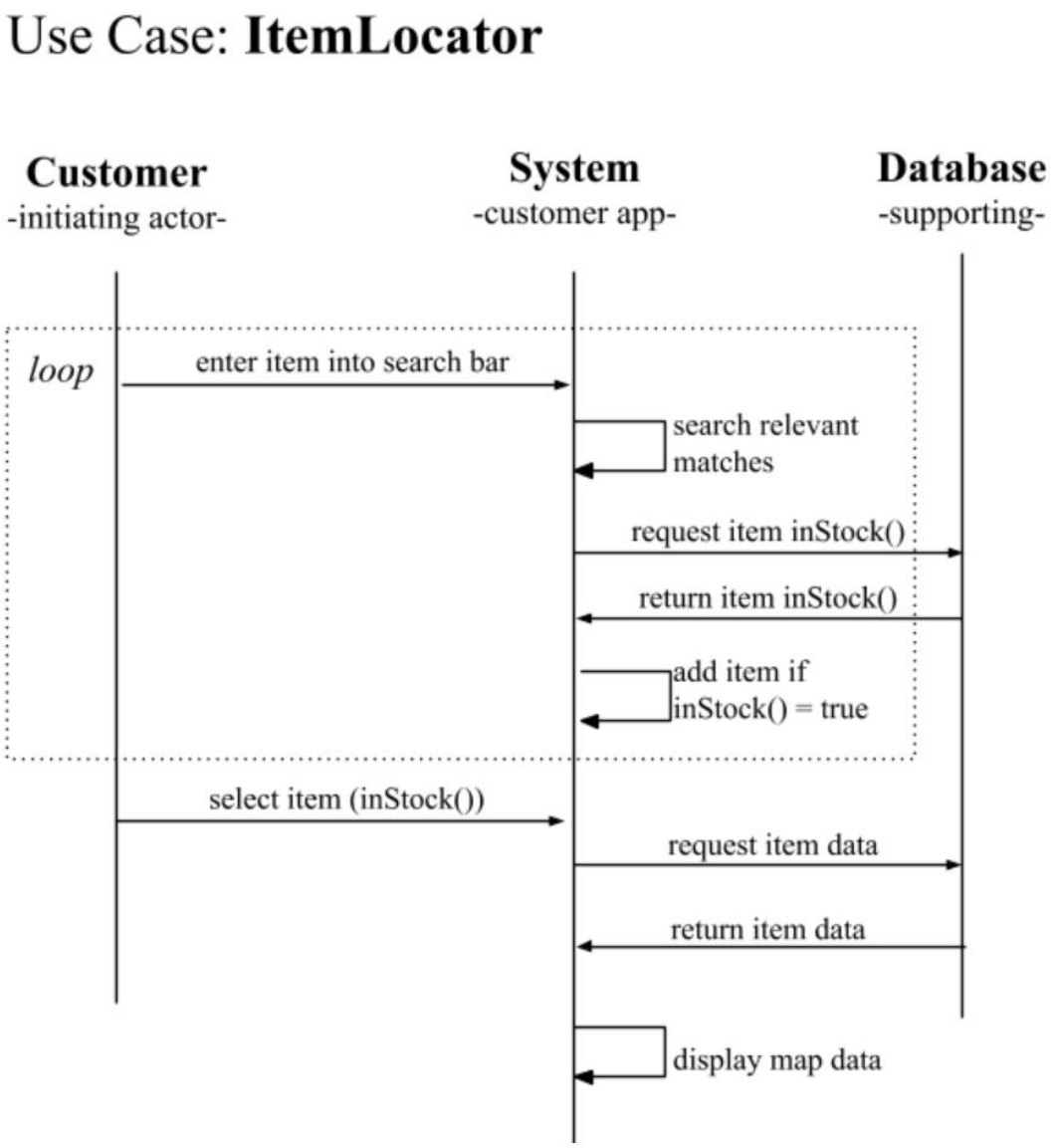


Figure 1-1

- **System** has a high coupling because it has to communicate to the database and the customer.
- **Database** is an Expert Doer because it has knowledge of an item inStock and all the information on an item.
- **Customers** have low coupling because the system makes it easy for customers to communicate.

Design patterns:

- **Decorator pattern** between customers and customer app, so that we can make changes to the customer app while not affecting customers experience. We can also have a pattern between the app and the database, therefore, we can organize modifications to the database better.
- **Command pattern** between system and database so, we can iterate through all the functions. For example, request item inStock(), return item inStock(), and request item data.``

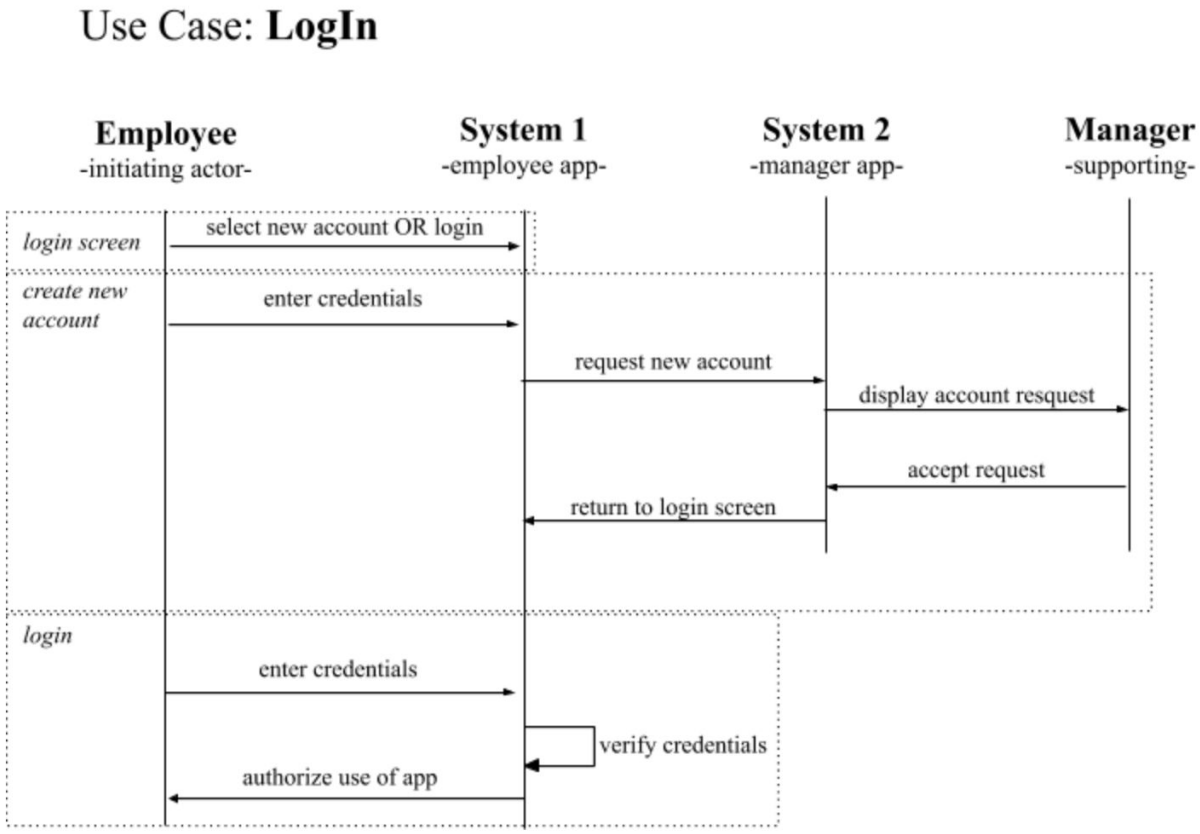


Figure 1-2

- **Employees** have high cohesion because they only have to type their correct credentials. (Single Responsibility Principle)
- **System 1** has low coupling because it only has to communicate with system 2.
- **System 2** has low coupling because the system does not have to send a lot of messages because system 2 shares some responsibility with system 1.

Design Patterns:

- **Decorator Pattern:** Ensures abstraction of login functionality between classes

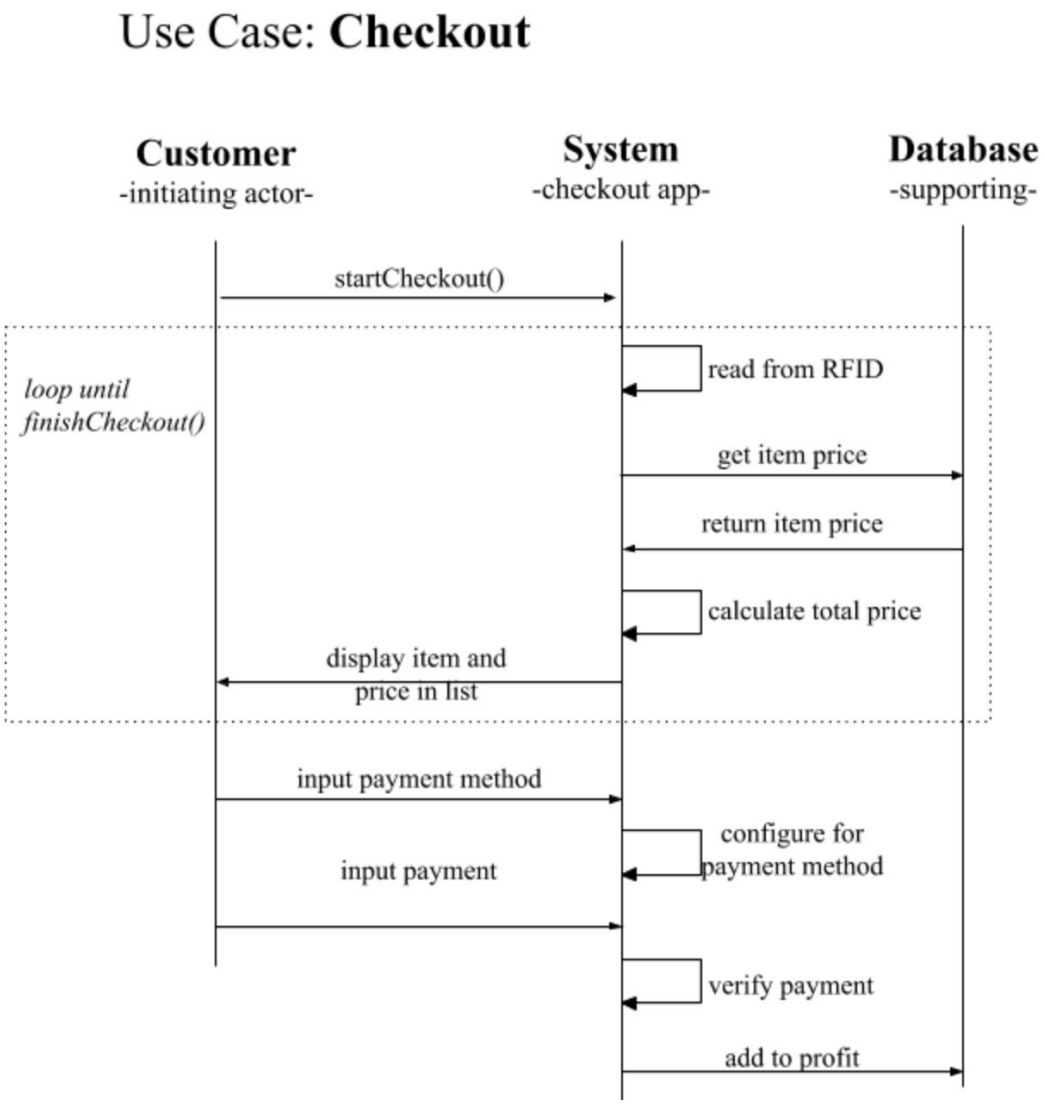


Figure 1-3

- **System** has a low cohesion as it performs many calculations
- **Database** is an Expert Doer because it has full knowledge of every item in the store, including it's price which it delivers to the system

Design Patterns:

- **Command pattern** between customer, app, and database so, we can just iterate through the functions like search OR scan item and request item price.

Use Case: **ReturnItem**

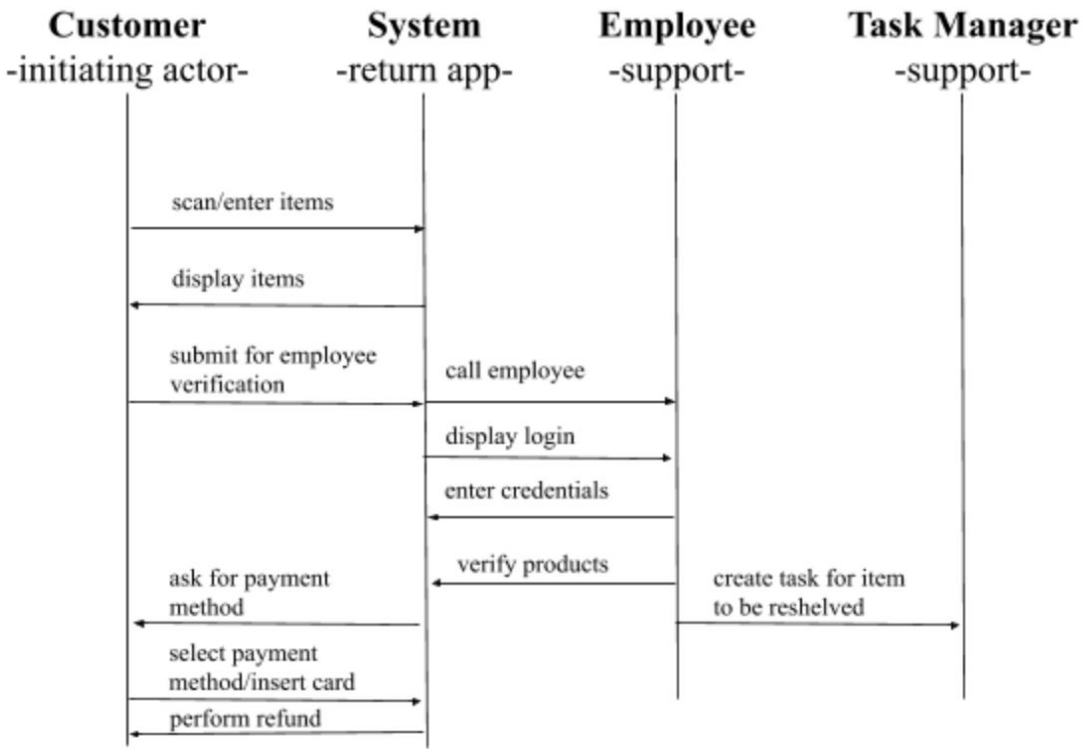


Figure 1-4

- **Customers and Employees** have high coupling because they have to constantly communicate with the system.
- **System** has high cohesion because it only has to work with the items on the receipt

Design Patterns:

- **Decorator Pattern:** Used for its pipe and filter style of separating functionality. It dissects unrelated responsibilities into separate classes. In such a way, the customer is unbothered with the working of the employee verification and the employee is focused only on the task of verification.
- **Command Pattern:** Used to make fetching and recording of item details easier via command line arguments.

Use Case: **PriceChecker**

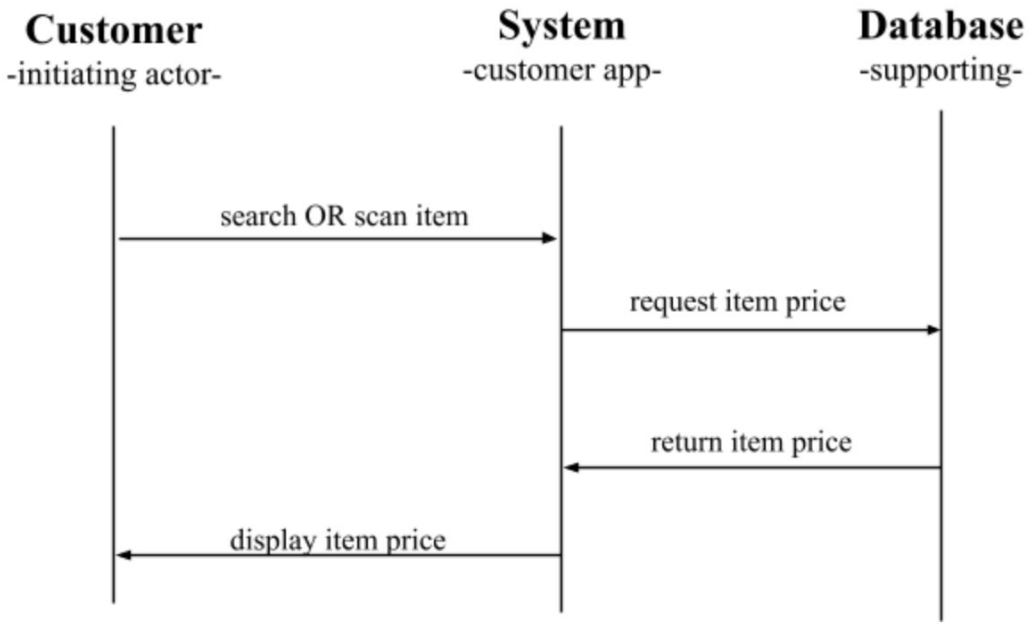


Figure 1-5

- **Each object** has high cohesion as they have only one responsibility.
- **Database** is an Expert Doer because it has full knowledge of every item in the store, including its price which it delivers to the system

Design patterns:

- **Command pattern** between customer, app, and database so, we can just iterate through the functions like search OR scan item and request item price.

Section 8: Class Diagram and Interface Specification

Section 8.1: Class Diagram

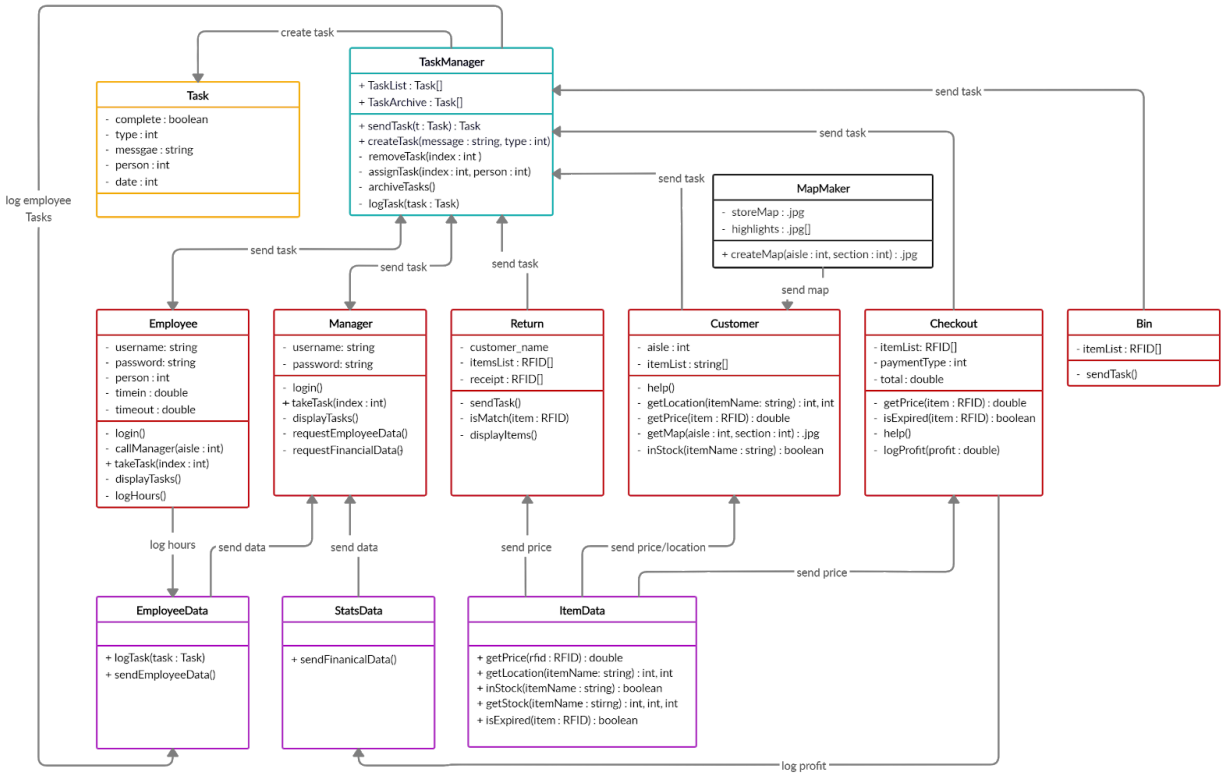


Figure 8.1-1

Section 8.2: Data Types and Operation Signatures

As our code is not class based we have added in the main local variables and functions for each application.

Task Board/Manager

Task[] TaskList // list of all Tasks obtained from querying the databse
sendTask(Task t) // sends a task to a controller type (employee or manager)
createTask(string name, string description)
// receives type and message info from one of the apps controllers
// creates Task based on info and adds to database
removeTask(int taskID) // removes a task
updateTask(int taskID, string name, string description, string state, int employeeID, datetime timeCreated, datetime timeCompleted)
// updates a task based on the parameters that need to be changed

Analytics

takeTask(int index) // assigns Task at index to the manager
displayTasks() // display all tasks on screen with buttons to takeTask()
requestEmployeeData() // requests and displays detailed table of employee data
requestFinancialData() // requests and displays detailed table of financial data

Return

int productID // ID of the transaction (for identification)
int productPrice // date of the transaction (for identification)
String productDescription //reason for the customer's return of the item
RFID[] itemsList // array of all items scanned/entered
UIController() // displays all scanned/entered items on screen
employeeLogin() //uses employee login feature
sendTask() // sends Task for an item to be reshelved

Shopping List

RFID[] itemList // list of items RFIDs
search(string name) //searches for an item
add() //adds the selected item
delete() //deletes the selected item

Checkout

RFID[] items // list of items scanned by RFID scanner

int paymentType // payment type either cash (0) or credit (1)
double[] totals // total price of items to be purchased
sendTask() // sends task for an item to be reshelved

Price Checker
RFID[] itemList // list of items scanned by RFID scanner
sendTask() // sends task for an item to be reshelved

ItemLocator
getPrice(RFID rfid) // returns price of scanned item
getLocation(string itemName) // returns aisle# and section# of item
inStock(string itemName) // returns true if item is in stock
getStock(string itemName) // returns number stock in total, shelved, and in storage

Section 8.3: Traceability Matrix

Domain Concepts	**Software Classes**										
	StatsData	ItemData	Task	Task Manager	Return	Checkout	Customer	Manager	Employee	Bin	MapMarker
Price Holder		X				X	X				
Stock Manager		X						X			
Item Locator		X					X				
Map Maker											X
Employee Database									X		
Financial Database	X							X			
Task Manager			X	X	X				X	X	
Teremial				X	X	X					X

Section 8.3 Design Patterns

Type	Pattern	System	Description
Creational	**Prototype**	**Returns**	A designated style was given to implement the items into the list, and this prototype was followed by search and replace methods to add the items into the list
Structural	**Adaptor**	**Returns**	The program was written with different controllers, one for UI, one for the mathematical functions, and one adapter to tie the two functions together
Behavioral	**Iterator**	**Returns, Checkout**	The iterator was called in order to sum the total of all the items in the list and to return the total value for checkout/return
Creational	**Builder**	**Returns**	A builder method was used in the UIController of the Returns System to build items on the list as they were entered/scanned as data into a displayable list entry
Behavioral	**Command**	**All Systems**	Because all of our systems do not directly interact with the database, the system interacts with a web page that in turn queries a call to the database to pull/edit items. (e.g. a user can create a new task, but this does not directly add it to our database, it call our code with then creates the value in the database with the value the user provided)

Section 8.4: Object Constraint Language (OCL) Contracts

Our Project is not exactly class based so we have decided that we are going to treat each separate module as class.

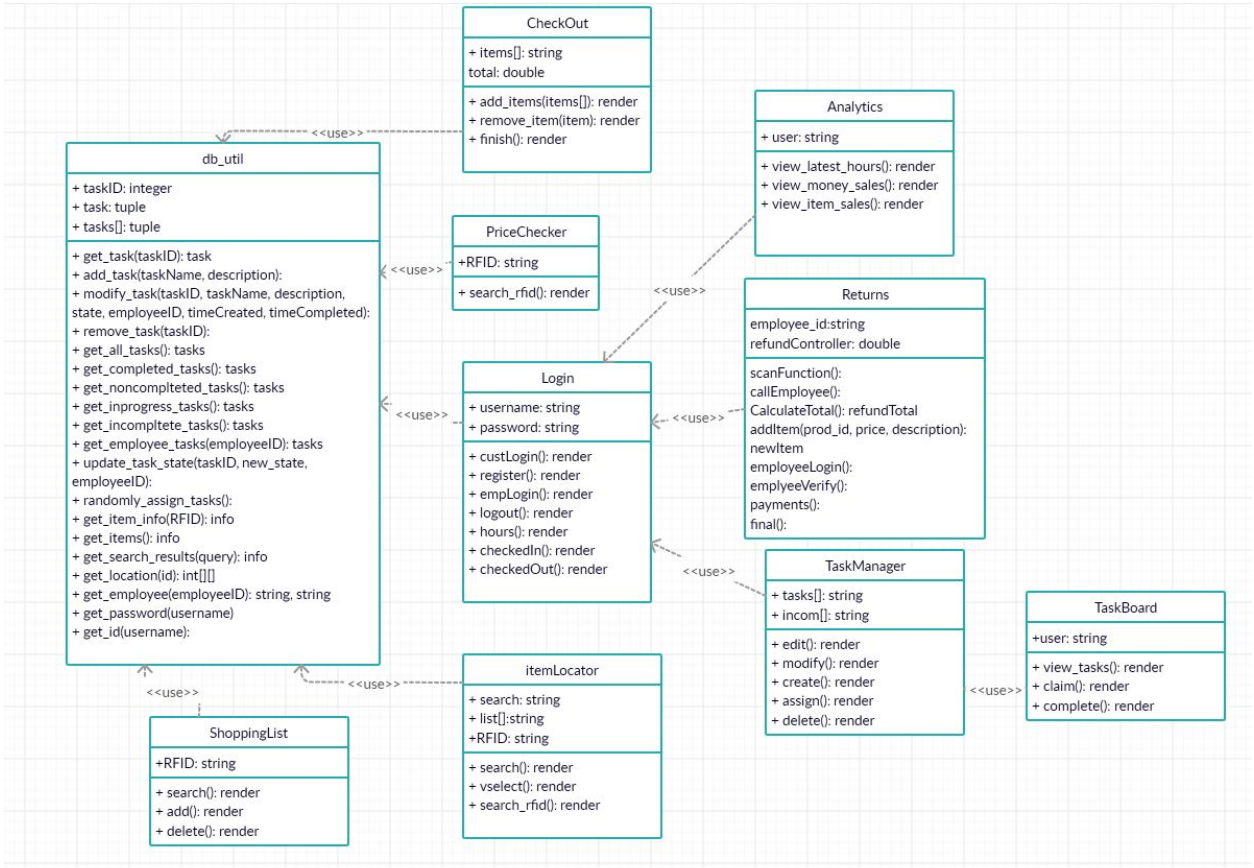


Figure 8.4-1

Section 9: System Architecture and System Design

Section 9.1: Architectural Styles

This system is a combination of the Backboard System and Database-Centric Architecture because communication typically occurs between one application and the database. The Customer Application, Employee Application, Manager Application, Checkout Terminal application, and Returns Terminal application do not generally directly interact with each other. Rather, they usually query a database to obtain the necessary information.

The Customer Application typically queries the database. Item search and price checking queries the database to obtain information about the price and/or location of the item. For customers logged in on their smart device, the customers would also query the database for information on items to be added to their shopping list; information about items on their shopping list may be saved on the user's device to allow the user to check their shopping list even when offline. The Employee Application will query the database for a task board. The Manager Application will query the database to retrieve information regarding employee information and accounting information.

Section 9.2: Identifying Subsystems

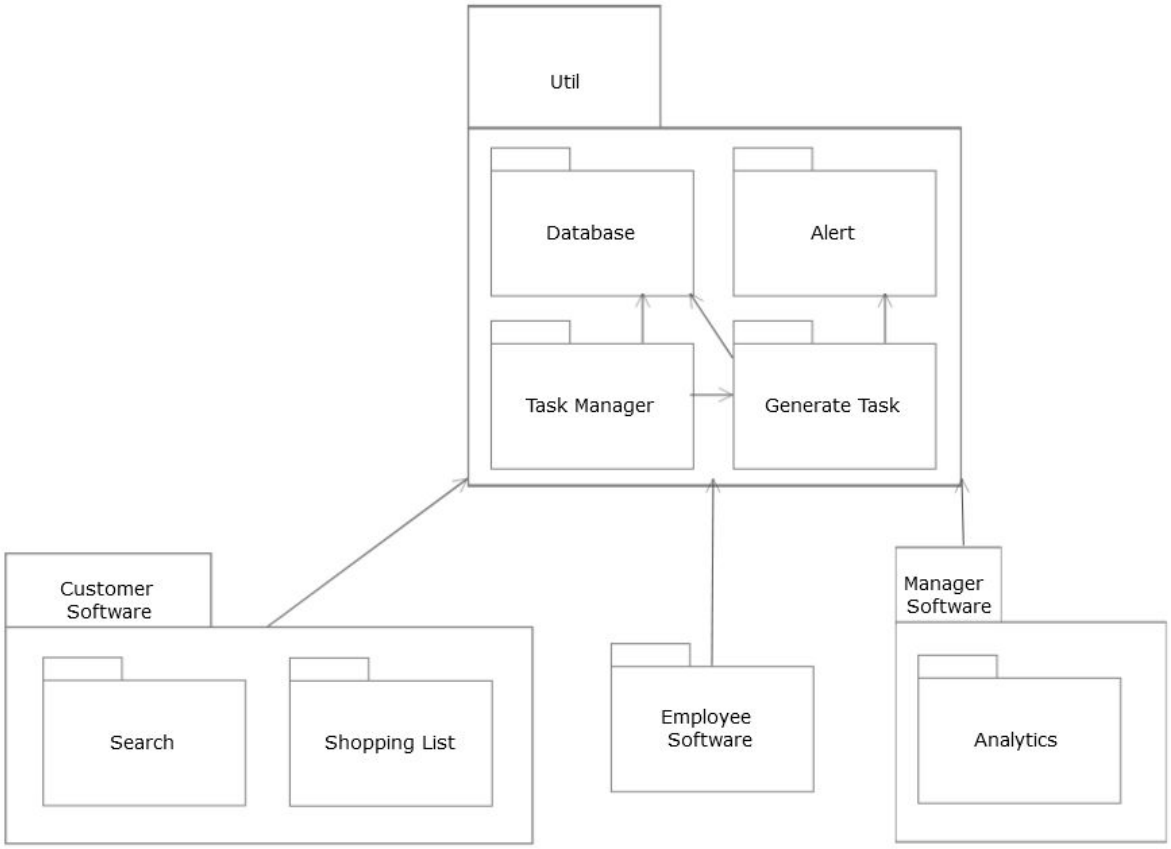


Figure 9.2-1

The util software package will control access to the database and manage the taskboard. This package interacts with other packages and contains functions that are used by multiple packages so as to reduce redundancy.

The customer software package will be able to search for items and manage a shopping list. It interacts with the util package to access the database to perform those actions. The customer software package also interacts with the util package to generate tasks, such as when a customer removes an item from their cart to be returned to the shelves or when a customer requests assistance.

The employee software package will be used to view the task board managed by the util package. When a task is generated by another package, the util software package will send an alert that will be received by the employee software package.

The manager software package will be used to both create and view tasks that are managed by the util package as well as query the database and perform analytics such as sales figures and employee task completions.

Section 9.3: Mapping Subsystems to Hardware

The system will run on multiple devices. There will be a Customer Application, an Employee Application, and a Manager Application.

The Customer Application will be run on either (1) Customer Assistance Terminals which exist around the store or (2) a customer's own smart device. Customer Assistance Terminals contain a tablet which will be running the Customer Application.

The Employee Application and the Manager Application will run on smart devices for employees and managers respectively. Grocery stores will determine whether the smart devices will be provided to employees and/or managers for usage in the store or whether the smart devices should be the personal smart devices of employees and managers.

Section 9.4: Persistent Data Storage

The system needs to save data regarding employees and stock past a single execution of the system. The system will be stored in a relational database using mySQL. For this project, the database shall be hosted on an AWS server.

A sample database was created so that the system could be illustrated. Some tables were created with the use of databases[6], random name generators[7], and other random generators[8].

items(*name*, *brand*, *type*, *RFID*, *price*)

The **items** table contains information about all products that could potentially be sold in the store. The *RFID* column is the primary key for this table since the RFID for each product is unique. The database should contain at most 4,294,967,296 tuples so as to ensure unique RFID numbers.

Column Name	**Description**	**Type**
name	This is the name of the product.	VARCHAR(65535)
brand	This is the name of the brand that produces the product.	VARCHAR(65535)
type	This is the type of product/category under which the product falls. This	VARCHAR(65535)

	will be used to help identify where the product should be located in the store.	
RFID	This is the number represented by the RFID tag that should be attached to the product when in store.	INT UNSIGNED
price	This is the price of the product in USD.	DECIMAL(10,2)

The sample database used contains 10,000 tuples for the **items** table. The Open Grocery Database Project[6] brands database was used for *name* and *brand* columns. Random integer number generation[8] was used to create a list of numbers between 10 and 21 which was replaced with a type depending on the number (ex. every item assigned 10 was of type Electronics). Random integer number generation[8] was used to create RFIDs for each item. Random integer number generation[8] was used to create prices, and ".99" was appended to the end of each generated integer.

stock(*itemRFID*, *amt*, *expirationDate*, *location*)

The **stock** table contains information about which items are in stock within the store and the backroom. There may be multiple tuples in the database with the same *itemRFID*. The first case is that the *expirationDate* of the tuples are all unique. This is to ensure that items are still shelved, but expired items will be removed. The second case is that one tuple has a *location* in the store while the other tuple has the *location* of BackRoom. This is to separate the items that are in stock in the store and the items that are in stock in the backroom.

Column Name	**Description**	**Type**
itemRFID	This is the RFID of the product.	INT UNSIGNED
amt	This is the amount of the product at *location*.	INT UNSIGNED
expirationDate	This is the expiration date of the item. This may be empty for some items, which do not expire.	DATETIME
location	This is the location of the product in the store. This includes the backroom.	VARCHAR(65535)

The sample database used contains 1,998 tuples for the **stock** table. This table is primarily generated based on the *RFID* and the *type* attributes of the **items** table. A CSV of 999 random numbers (1_code/Database and Database Generation/Generation/random_items_numbers.csv file in the GitHub repository) was generated with numbers between 1 and 10,000 using a random integer number generator[8]. Those random numbers were used to determine which row in the **items** table should be used (for the *itemRFID* attribute). The generated table has two tuples for

each item—a tuple for the item in the back room, and a tuple for the item on the shelves. The *amt* attribute is randomly generated; the tuples in the back room have an *amt* attribute between 0 and 2,000; the tuples on the aisles have an *amt* attribute between 0 and 1,000. The *expirationDate* attribute is a randomly generated date and time between January 1, 2020 at 0:00 and December 31, 2020 at 23:59. The *location* attribute is generated based on the *type* of the item in the **items** table as follows:

Location	**Item Type**
Aisle 1	Electronics
Aisle 2	Bath
Aisle 3	Beauty
Aisle 4	Health
Aisle 5	Drink
Aisle 6	Meat
Aisle 7	Grain
Aisle 8	Dairy
Aisle 9	Frozen
Produce	Fruit, Vegetable
Back Room	<any>

To see more clearly how this table is generated, please view the 4_data_collection/Data Generation/generator.py file in the GitHub repository.

employees(*lastName*, *firstName*, *ID*, *role*)

The **employees** table contains basic information about employees. The *ID* is the primary key for this table since each employee's employee ID should be unique. The database should contain at most 4,294,967,296 tuples so as to ensure unique employee numbers.

Column Name	**Description**	**Type**
lastName	This the last name of the employee.	VARCHAR(65535)
firstName	This is the first name of the employee.	VARCHAR(65535)
ID	This is the employee's employee ID.	INT UNSIGNED

role	This is the role of the employee.	SET(Employee, Manager)

The sample database used contains 60 tuples for the **employees** table. The *lastName* attribute is generated with a random last name generator[7]. The *firstName* attribute is generated with a random first name generator[7]. The IDs were randomly arranged numbers from 1 to 60. Random integer numbers from 1 to 4 were generated using a random integer number generator[8]. Employees with a certain number for their role (ex. 2) were assigned the "Manager" role, and the rest were assigned the "Employee" role. This was done to ensure that there were more employees than managers.

hours(*employeeID*, *day*, *checkIn*, *checkOut*, *dayHours*)

The **hours** table contains information about employees' hours per day.

Column Name	**Description**	**Type**
employeeID	This is the employee's employee ID.	INT UNSIGNED
checkIn	This is the time when the employee checked in.	DATETIME
checkOut	This is the time when the employee checked out.	DATETIME

The sample database used contains 999 tuples for the **hours** table. The hours table is generated based on the **employees** table. For each tuple, a random employee ID is chosen for the *employeeID* attribute. In the original generation, a day was randomly generated between January 1, 2020 and December 31, 2020. A check in time was generated between 1:00 and 20:59. To ensure that check outs were on the same day as the check in, the checkout time was generated based on the check in time and ensured to be before 0:00 of the next day; some checkout times were generated as NULL. The day column was manually merged with the checkIn and checkOut columns. The dayHours column was calculated in the original generated but was later removed because it could be calculated with an SQL query. To see more clearly how this table is generated, please view the 4_data_collection/Data Generation/generator.py file in the GitHub repository.

tasks(*taskName*, *description*, *taskID*, *state*, *employee*, *timeCreated*, *timeCompleted*)

The **tasks** table contains information about tasks that have been created for employees to complete. The *taskID* is the primary key for this table.

Column Name	**Description**	**Type**

taskName	This is the name of the task.	VARCHAR(65535)
description	This is a description of the task.	LONGTEXT
taskID	This is the number of the task in the system and is used by the system to identify a specific task.	INT
state	This is the state of the task, describing whether the task is untaken (Incomplete), currently being completed by an employee (In Progress), or completed by an employee (Complete)	SET(Incomplete, In Progress, Complete)
employeeID	This is the employee ID of the employee who either is completing the task or has completed the task.	INT UNSIGNED
timeCreated	This is the time that the task was created.	DATETIME
timeCompleted	This is the time when the task was completed. This may be blank for some tasks.	DATETIME

The sample database used contains 20 tuples for the **tasks** table. This table was manually generated. The *employeeID* attribute was manually selected (with an attempt to be relatively random), and all other attributes except for the *taskID* attribute were manually created.

sales(*transactionID*, *time*, *totalPaid*)

The **sales** table contains information about transactions that were made in the grocery store. The *transactionID* and *time* are the primary keys for this table.

Column Name	**Description**	**Type**
transactionID	This is the transaction ID of the transaction.	INT UNSIGNED
time	This is the time that the transaction was made.	DATETIME
totalPaid	This is the total amount paid by the customer.	DECIMAL(10,2)

The sample database used contains 99 tuples for the **sales** table. The time of the first transaction is randomly generated. For each day, the *transactionID* attribute increments starting from 1. To determine the time of the next transaction, whether the day should be incremented is randomly determined. If the day is incremented, it uses the same time as the previous transaction; otherwise, the time is incremented between 1 and 1,000 minutes (which can result in the day being incremented). The **sales** table is generated at the same time as the **transactions** table, and information based on the items in the **transactions** table (i.e. the *price* attribute in the **items** table based on the *item* attribute in the **transactions** table, which is the item's RFID; and the *amt* attribute in the **transactions** table) is used to calculate the *totalPaid* attribute. The *totalPaid*

attribute is calculated by summing the products of the prices and amount for each time, and then multiplying that sum by 1.07 for tax. To see more clearly how this table is generated, please view the 4_data_collection/Data Generation/generator.py file in the GitHub repository.

transactions(*transactionID*, *item*)

The **transactions** table contains information about what items were bought with each transaction.

Column Name	**Description**	**Type**
transactionID	This is the transaction ID of the transaction.	INT UNSIGNED
time	This is the time that the transaction was made.	DATETIME
item	This is the RFID of the item that was bought in the transaction.	INT UNSIGNED
amt	This is the amount of the item that was bought in the transaction.	INT UNSIGNED

The sample database used contains 1,595 tuples for the **transactions** table. For each *transactionID* and *time* combination in the **sales** table, an amount of items (from 1 to 30) is generated for the sale. Each item is randomly and uniquely selected for each sale based on randomly and uniquely generated numbers that select a row in the **items** table. The *amt* attribute is a number between 1 and 10. To see more clearly how this table is generated, please view the 4_data_collection/Data Generation/generator.py file in the GitHub repository.

logins(*username*, *password*, *accountType*)

The **logins** table contains information about login information for usage of the Customer Application, the Employee Application, and the Manager Application on smart devices.

Column Name	**Description**	**Type**
username	This is the username of the account.	VARCHAR(65535)
password	This is the password that is used to log into the account.	VARCHAR(65535)
accountType	This is the type of the account, which will determine which application the user can access.	SET(Employee, Manager, Customer)
ID	This is the user's employee ID if application. It is NULL otherwise.	INT UNSIGNED

The sample database used contains 70 tuples for the **logins** table. The *username* and *accountType* attributes were manually generated. For the first 60 tuples, the *username* and *accountType*

attributes were generated based on the *firstName*, *lastName*, *role*, and *ID* attributes in the **employees** table. The *username* was manually generated for the last 10 tuples, and the last 10 tuples had "Customer" for the *accountType* attribute and NULL for the *ID* attribute. The *password* attribute was randomly generated using a random alphanumeric string generator[8]. The first 60 tuples had passwords of length 10, and the last 10 tuples had passwords of length 8.

shoppingLists(*username*, *item*)

The **shoppingLists** table contains information about customers' shopping lists.

Column Name	Description	Type
username	This is the username of the account.	VARCHAR(65535)
item	This is the RFID of the product in the customer's shopping list.	INT UNSIGNED

The sample database used contains 178 tuples for the **shoppingLists** table. This table is generated using the last 10 tuples of the **logins** table, which are all customer accounts, and the **items** table. To see more clearly how this table is generated, please view the 4_data_collection/Data Generation/shoppinglistgenerator.py file in the GitHub repository.

Section 9.5: Network Protocol

The system shall use the Python mySQL connector. This choice was made because (1) a mySQL database will be used, and (2) the system will be coded in Python.

Section 9.6: Global Control Flow

The system is typically event-driven. Events are typically initiated by a customer or manager. Customers may initiate events to search for an item, to check the price of an item, to remove an item from their cart, or to purchase items; these will interact with only the database. Customers may also initiate events to request for help, which will interact with both the database and the Employee Application. Managers can create tasks, which will interact with both the database and the Employee Application. Rather than specifically waiting for an event, the events directed towards the Employee Application will work similarly to interrupts; in other words, the events are asynchronous.

Section 9.7: Hardware Requirements

The system depends on screen display on tablets and smart devices, RFID sensors, an Internet connection, and either (1) database disk storage on AWS or (2) an on-site database disk storage and a local network connection. The screen display will display the Customer Application, the Employee Application, and the Manager Application. The RFID sensors are used to identify items for purchase, price checking, or returns.

The database disk storage on AWS will be used to store persistent data; grocery stores may opt to have a backup disk storage copy of the database available on site in case of Internet outage. As the database exists on AWS, an Internet connection will be required to access it. If an on-site database disk storage is used in lieu of an AWS database, then an Internet connection is still necessary for customers to access and update their shopping list, and a local network connection could be necessary to access the on-site database.

Section 10: Data Structures

The data structures used in the program are simple because the backbone of the program is a database. For the checkout application, A linked list was used to store just the item and the price after transferring from the database. This was used to mimic the usage of the shopping cart in order to allow easy data manipulation. A linked list is best suited for this task so items and tasks can be removed out of order by searching through the list and removing. It is also useful in showing multiple items of the same type in the display functionality of the application. The main criteria used is flexibility and not performance because our program is not doing enough computations to require performance considerations.

Section 11: User Interface Design and Implementation

Section 11.1: Preliminary Design

Customer Assistance Terminal

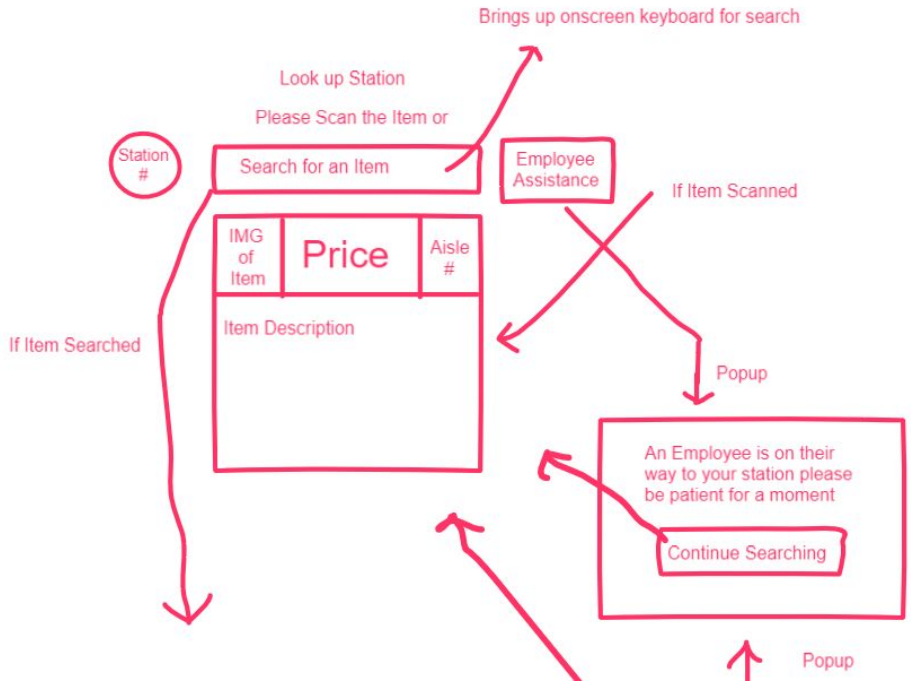


Figure 11.1-1

For the lookup station at a Customer Assistance Terminal, there are multiple paths. The first path is to scan an item in order to check the price as shown in Figure 11.1-1. This is a quick RFID scan and then the app displays the relevant information. If assistance is needed then the user would press the needs assistance button. This will send an alert to an employee and that employee who gets the task up will receive a priority task added to their task list.

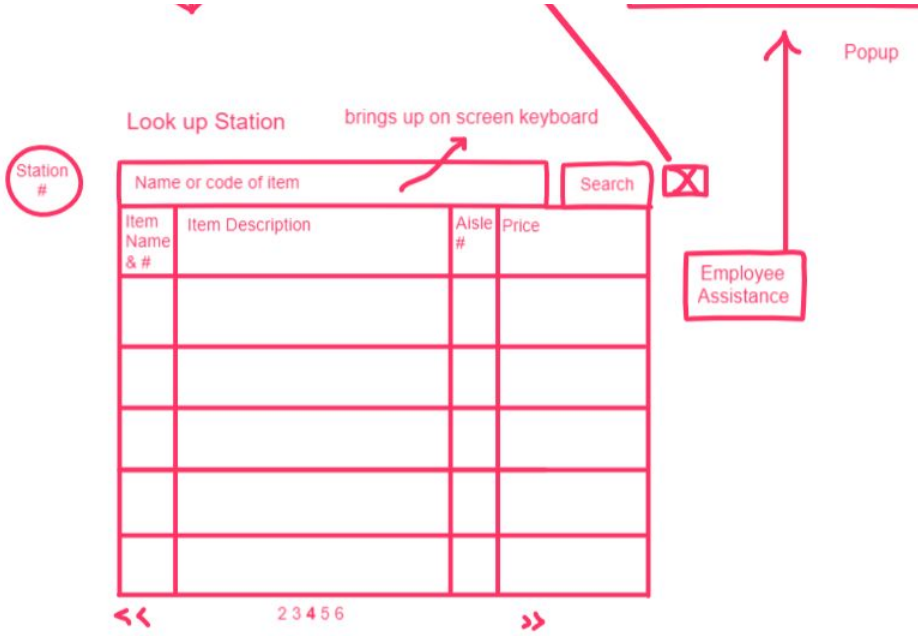


Figure 11.1-2

The other path is to use the search function in order to find an item's location in the store or to compare similar items as shown in Figure 11.1-2. This is done by clicking the search bar and then searching through keyword or item number and then hitting enter. The program will then show you the search results. There is also an employee assistance button on this page that will act the same as the other button.

Employee Application

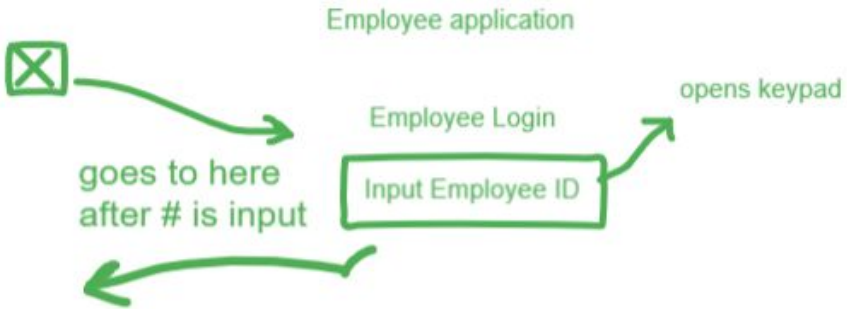


Figure 11.1-3

For the Employee application the first step is to input their employee ID. The display is shown in Figure 11.1-3. This will then allow them to access 4 separate functions: clock in/clock out, check hours, check schedule, and task list. These are shown below in Figure 11.1-4.

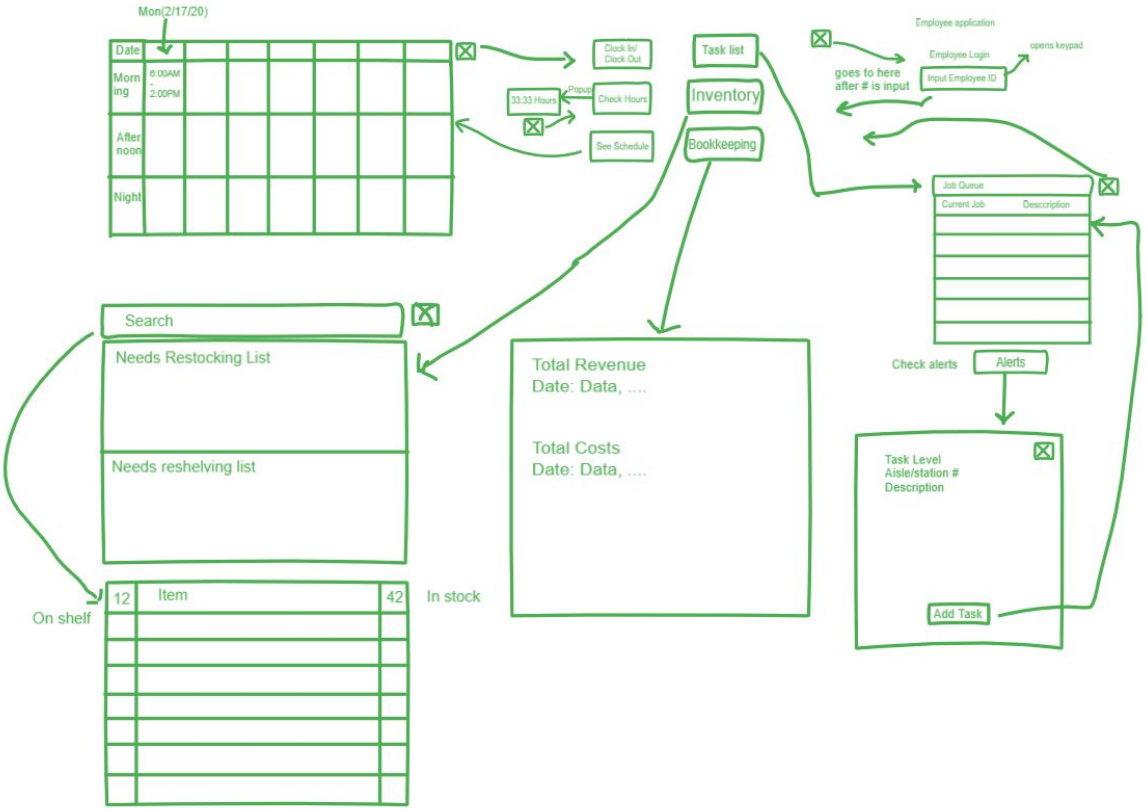


Figure 11.1-4

The clock in/clock out button will track the employees hours in the timesheet database. The check hours buttons will activate a pop up with the amount of hours worked that week, but for a manager, it will show the total hours worked for every employee. The check schedule button will show when an employee is working that week and for the manager will show who is working in any given time slot. Then on the task list screen an employee can see what their current tasks are and the priority levels of these tasks. It also has the alert button so that if they are looking for a task or a priority task shows up it can get allocated properly.

Any manager has some manager specific apps: inventory and bookkeeping. The inventory button allows a manager to see the inventory of the store and allow the allocation of tasks. First the manager would click the button and then they would see what needs to be restocked and what needs to be reshelved. Then they would be able to search for the specific stock of an item through the search bar. Finally, the bookkeeping button allows a manager to see the total revenue and costs of the store.

Customer Application

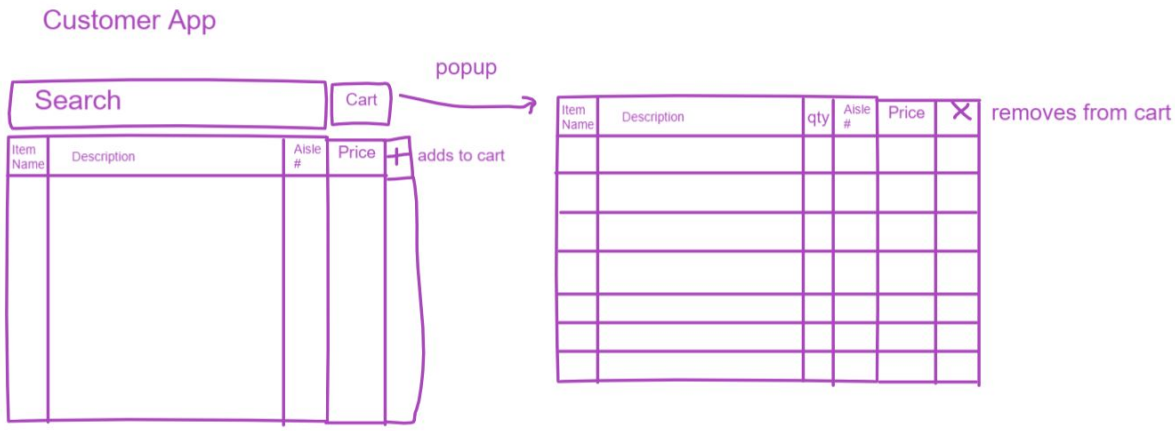


Figure 11.1-5

For the customer application we have a simple app that has the same search capability of the in store tablets but for a specific customer as seen in Figure 11.1-5. It allows the user to search items and add them to their cart and remove them as necessary. It also gives the ability to see the aisle where the item is located and check the price of said item.

Checkout Terminal

Figure 11.1-6

As stated in the project problem statement, the checkout system should be mostly employee involvement free, unlike current self-checkout stations at supermarkets. The user is initially greeted with a direction to place their bag in the RFID scanning area; this is shown in Figure 11.1-6. This area will scan the RFID of all the products in a user's bag and then will automatically go to the Cart Screen.

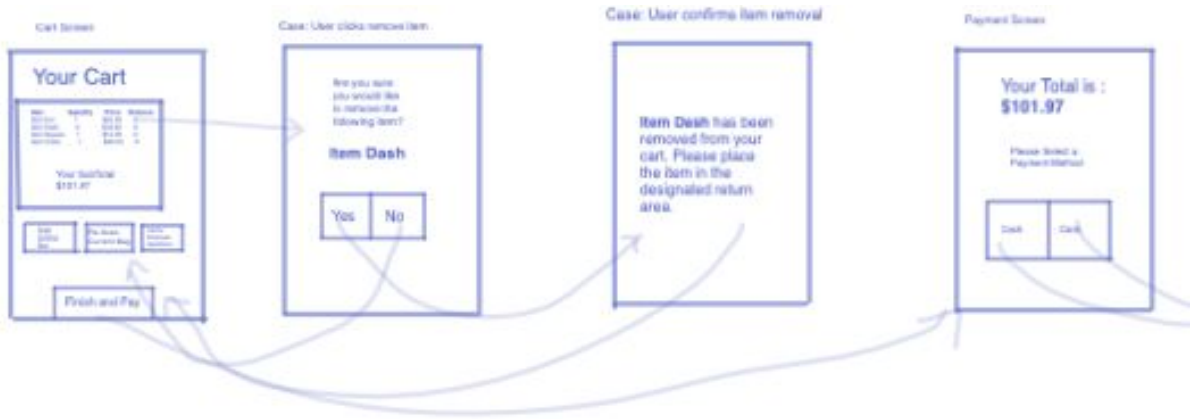


Figure 11.1-7

This screen displays all of the items that the scanner has recognized and displays the name, quantity, and price of the item along with the option to remove any specific item; the screen

should look like the leftmost screen in Figure 11.1-7. The subtotal of all the products scanned is also indicated to the user. Once they see this, they have options to either scan another bag, re-scan the current bag if there are any errors present in the RFID scanning, call an employee for assistance for more specific and unexpected problems, or finish and pay.

If the user selects to scan another bag, they will be directed back to the welcome screen and their new bag's items will be added to the current cart total. If they wish to rescan the current bag, all items in the cart will be deleted and they will be led back to the welcome screen. Calling an employee for assistance will notify an employee of what system is requiring their assistance. If they select finish and pay, they will be directed to the payment screen.

It was stated earlier that the user was given options to remove specific items from their cart; this topic shall be further expanded on. If the user elects to remove an item, they are taken to a page which asks them to confirm removal of the item; if they elect to then not remove the item, the item is not removed from the cart and they will go back to the cart screen.The associated screens are the second and third screens in Figure 11.1-7.

If a user elects to remove an item from the cart, then the item is deleted from the cart, and they are led to another page which directs them to place the removed item into a designated area as specified earlier in this report, and afterwards they are taken back to the cart screen.

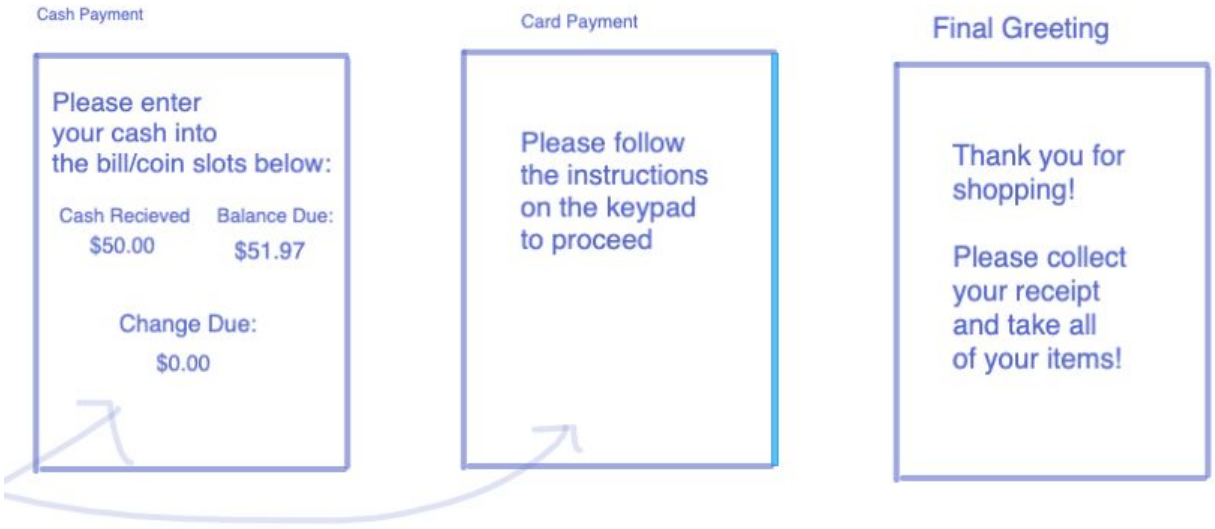


Figure 11.1-8

The case when the user selects the finish and pay button shall now be examined. When the user clicks this, then their subtotal is calculated with taxes and displayed on a screen as shown in the rightmost screen in Figure 11.1-7. On the same screen, they are asked whether they would like to make a card payment or a cash payment.

If they select cash payment, they are taken to a cash payment screen and directed to enter bills/coins into the register's designated slots. This cash payment screen is shown in the leftmost screen in Figure 11.1-8. The system will tell them the amount of money that has been processed, the balance that is due, and if they have paid more than their total, the change that is due to them before it takes them to the final screen.

If the user selects card payment, then they are directed to follow the instructions on the given keypad as they would for all card payments. This will be shown like on the middle screen in Figure 11.1-8. After the user is done scanning and paying for all of their items, they will see the final greeting screen that thanks them for shopping and remind them to collect their receipt and all of their items.

Returns Terminal

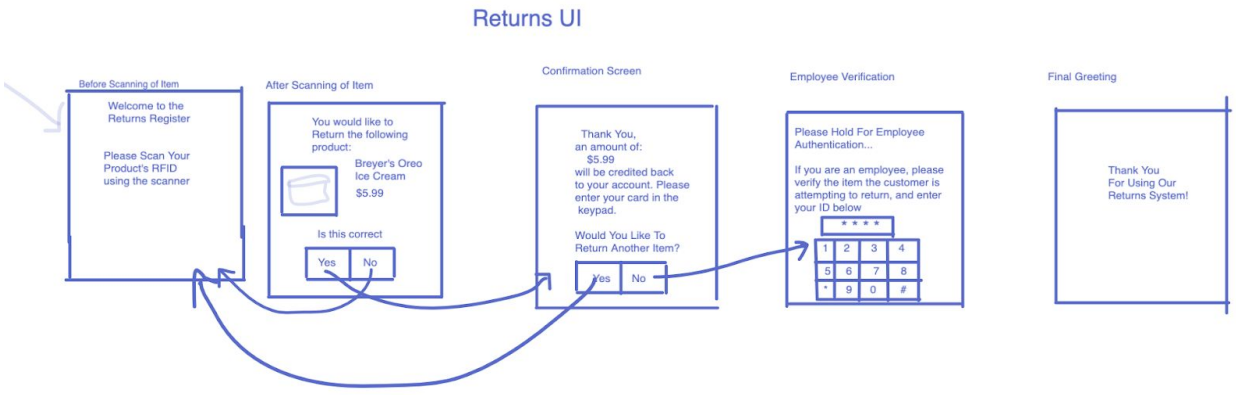


Figure 11.1-9

The process of the return terminal shall be explained. The screens involved in this case are shown in Figure 11.1-9. The user will first be prompted to scan the item they wish to return using the RFID scanner. The scanner will scan the item and then display the item's name, weight, price, and in some cases, an image of the item. The user is then asked to verify whether the scanner has scanned the right item, and if not they will be asked to rescan it. If the product is scanned correctly then the user will be asked whether they would like to scan another item. If they select yes, the process restarts, and the item is added to a list, if not, then they are asked to hold for employee authentication as mentioned earlier in this report.

Although having an independent self-checkout system would be greatly beneficial, preventing cases of possible user abuse of the system is more important as it could end up costing the company a loss. The employee verification is put in place so that users can't return an empty box or scan a proxy RFID. The employee verification page asks for an employee to enter their ID, which like passwords, is hidden from the customer, and once the employee ID is confirmed, then the customer is prompted to a confirmation screen saying that the amount will be credited back to the account for which they enter their card in for, which then automatically changes to the final greeting screen.

Section 11.2: User Effort Estimation

User wants to find what aisle cereal is in: The user will find a lookup station and click on the search bar to bring up the keyboard. Then they will input the type of cereal and hit search. The program will show search results along with a short description, price, and aisle for each. 2 clicks, 1 keystroke.

User needs assistance after price lookup: User scans the item at lookup station(1 action). Then the User pushes the employee assistance button(1 click). An employee then gets a priority task notification on their app that a user needs assistance at station 00. 2 actions, 0 keystroke

Employee wants to check their current task: The employee opens their app and inputs their ID #(1 keystroke). Then they would click the task list button(1 click). 1 clicks, 1 keystroke

Employee wants to check current weekly hours: The employee opens their app and inputs their ID #(1 keystroke). Then they would click the check hours button(1 click). 1 clicks, 1 keystroke

Manager wants to check current stock of an item: The manager will log in to the app with his employee ID number(1 keystroke) and then he will choose the inventory button(1 click). He then uses the search bar to find the stock on the shelf and in the storeroom of said item(1 keystroke). 1 click 2 keystrokes

User wants to check out 2 bags of items: User places first bag in scanning area, clicks scan another bag (1 click), places second item in scanning area, clicks 'finish and pay' (1 click). User pays with a card so that it is selected (1 click), and the user proceeds to use the keypad to finish the transaction. (overall 3 clicks)

User wants to check out 2 bags of items, but wants to remove 1 item: User places first bag in scanning area, sees item for removal in the cart screen and clicks the 'X' for removal (1 click). They then click to confirm removal (1 click) and then select scan another bag for their second bag (1 click) then finish and pay (1 click) then pay with cash (1 click) and finish the transaction accordingly. (overall 5 clicks)

User wants to return 2 items: User scans first item and verifies that it is correct (1 click), then and user says they would like to return another item (1 click), verifies it is correct (1 click), then says they would not like to scan another item (1 click) and employees comes in to enter their information (1 keystroke). (overall 4 clicks, 1 keystroke)

Section 11.3: Changes

Some minor changes were made from the initial screen mockups that were developed for Report #1. When the screen mockups, processes, and responses were first developed, "ease-of-use" into account was not taken into account as much as was done with the later designs. Once the number of clicks it would take for the customer/employees to access a certain process was determined, it was discovered that there were some extraneous steps in the processes.

The current version of our Returns Terminal in Figure 5.1 as shown below will be examined:

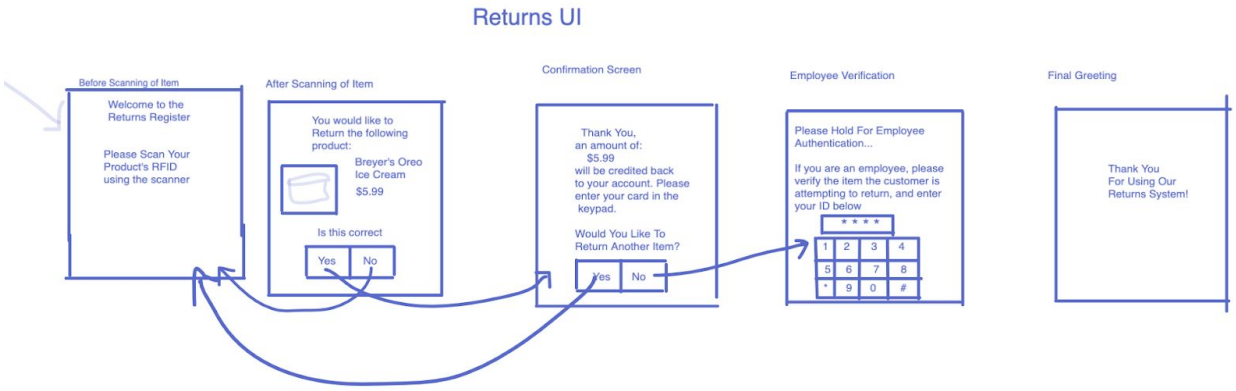


Figure 5-1

When the processes for the Returns system was initially drawn up, the employee verification screen came before the confirmation screen, and the employee would perform the confirmation. However, using a test case where the customer has to perform multiple returns, it was quickly discovered that the employee would need to enter in his or her verification ID each time for

every single item. Due to this, confirmation access was given to the user who selects whether or not they want to return another item, and once the user has returned all of their items, the employee can then enter in their verification ID once for all of the items as a summed return. This drastically reduced the number of clicks and proved to be a much more efficient process.

Apart from the returns terminal, ease-of-use was kept in mind while designing our other terminals, and that the initial version still proves to be the most efficient way of completing the required processes. Overall, all of the terminals were kept simple and provided a lot of detail along with limited keystrokes to the user to develop a user-friendly, guided system for the supermarket.

Section 12: Design of Tests

Test-case Identifier: TC-1 **Use-case Identifier:** UC-8 **Pass/Fail Criteria:** The test passes if the customer checkout is completed without any errors returned to the customer **Input Data:** Item barcodes, Customer Payment information, Remove item button	
Test Procedure: Step 1: Have the customer scan items, remove an item in the process, and then proceed to checkout.	**Expected Result:** For Step 1: The scanned items will all be in a list with their price and the calculator will return the total with the state tax. If the customer wishes to return an item, then the item will be removed from the cart, and a new total will be calculated. The customer's payment method will be asked and the process will be completed once the customer pays the balance.

Test-case Identifier: TC-2 **Use-case Identifier:** UC-2 **Pass/Fail Criteria:** The test passes if the item is located and the location of the item in the store is returned to the user. If the item is invalid, that information is also returned to the user. **Input Data:** Item's unique identifiers such as barcode, name, etc.	
Test Procedure: Step 1: Enter the barcode ID for an item that is in the store. Step 2: Enter a random barcode ID for an item that is not in the store (an invalid item)	**Expected Result:** For Step 1: The location of the item is presented to the employee/customer in a way that makes it easy for the user to locate in the store. For Step 2: The screen displays to the user that the item is invalid.

Test-case Identifier: TC-3 **Use-case Identifier:** UC-13 **Pass/Fail Criteria:** The test passes if the user goes through the entire returns process without returning any errors in the system. **Input Data:** Return item barcode, customer repayment information, Employee ID	
Test Procedure:	**Expected Result:**

Step 1: Have the customer return 1 item that is valid and have them go through the returns terminal. Step 2: Have the customer return an empty box that an employee identifies and uses the terminal to indicate this.	For Step 1: The customer goes through the returns terminal and returns the items that they want to. The employee verifies their returned items and enters in their Employee ID, and the employee will get the refund through their requested payment method. For Step 2: Same steps as Step 1, however, the employee will check the items and enter that item is invalid, and enter this in the Returns terminal when asked to enter their employee ID, and this will quit the Returns process.

Test-case Identifier: TC-4 **Use-case Identifier:** UC-15 **Pass/Fail Criteria:** The test passes if a valid employee ID is entered, and the correct working schedule is returned for the employee along with the employee wage. Using the schedule and wage, the system should calculate the employee salary. **Input Data:** Employee ID	
Test Procedure: Step 1: Enter a valid Employee ID in the terminal, then select calculate salary. Step 2: Enter an invalid Employee ID	**Expected Result:** For Step 1: This returns the employee's schedule initially, and when asked to calculate salary, the terminal uses the hours of the employee and the employee wage to calculate weekly salary and that amount is returned and displayed. For Step 2: This returns that this is not a valid employee ID, and refreshes to the home screen automatically in ~5 seconds.

Test Coverage Strategy

In terms of coverage, the code is being tested to the greatest degree through TC-1 and TC-3. These are the two terminals that customers will be interacting with when checking out and returning their items. To ensure sure that this process is flawless and contains no bugs, these designed test cases examine the majority of scenarios that would occur when customers are using this terminal.

Through TC-1, the checkout scenario is analyzed and tested using various items to check that adding a large amount of items does not slow down the system; if our program passes this test, then the customers should be able to buy a large sum of items with no problems. The process behind customers wanting to put an item back during checkout is also tested, and if the system passses this as well, one can be confident that the system can handle most customer checkouts.

Through TC-3, the returns process is analyzed and tested much like the checkout process was tested, but with an added step: employee verification. Much like checkout, it must be ensured that returning a large sum of items does not slow down the system or cause bugs, especially since the returns terminal is responsible for refunding the customer as well. There is also a test to make sure that the employee verification is working well, and through this, most if not all possible return scenarios are tested.

Integration Testing Strategy

The integral part of the program is the connection between the user and the database. Because this project has few systems, a Big Bang approach to integration testing should be sufficient. In order for the system to work every part would need to be there. As the user will be interacting with a GUI ,there needs to be a proper interaction between them. The program will have helper methods called by the GUI in order to call the database, and then the database will send the proper information back to the GUI so that it can be displayed. To test these functions, the database will be populated with sample values manually, and then the GUI will be used to access this data through the search function.

The actual testing will involve a check to make sure the GUI is actually calling the methods, followed by a check to make sure the methods are properly referencing the database, which finally is followed by checking to make sure the data is being retrieved by the database and implemented by the GUI.

For example, the integration of test case 1 would be tested by connecting all the parts together and then if there are errors using test markers in order to check if the program is properly contacting the database or not. In this specific case, that would be testing that the GUI is properly calling the method to remove an item and that the item is being removed from the cart.

Section 13: History of Work, Current Status, and Future Work

Section 13.1: History of Work

Work on the program was started much later than anticipated and completed close to the deadline for the first demo. A similar situation occurred for the second demo.

Work for the reports was completed approximately a day before the deadline, and edits were made the day of.

Section 13.2: Key Accomplishments and Current Status

Database Utility

The database is completely implemented. The example database contains all tables as described in Section 9.4. Functions have been created to query the database to obtain data from the database, change tuples in the database, create tuples in the database, and delete tuples in the database.

Task Manager

The core functionality of the task manager is implemented. Tasks can be viewed, edited, added, and deleted. It is integrated with the database to access and edit the tasks table. It is integrated with the login subsystem to allow only managers to access it.

Task Board

The core functionality of the task board is implemented. Tasks can be claimed and completed. It is integrated with the database to access and edit the tasks table. It is integrated with the login subsystem to allow only employees to access it and to obtain the employee ID of the logged in user.

Checkout

The core functionality of the checkout is implemented. Since there is no actual hardware, the scanning of items in the bagging area is simulated, and the items are added to the cart and can be seen on the screen. Items can be removed from the checkout list. It is integrated with the database.

Returns

The core functionality of the returns system is implemented. Items can be returned by RFID (simulating the RFID scanner that would be present) and price, and employees can approve the return. Employees can also be called for help. It is integrated with the database.

Price Checker
The core functionality of the price checker system is implemented. The prices of all items can be checked based on RFID. It is integrated with the database to check item prices.

Item Locator
The core functionality of the item locator is implemented. It is integrated with the database to obtain the item location.

Login
The core functionality of the login system is implemented. It is integrated with the database to check usernames and passwords. It uses session text to allow other subsystems to determine if a user is logged in.

Section 14: References

[1] Meyersohn, N., 2020. “Another Whole Foods competitor just bit the dust.” *CNN*. https://www.cnn.com/2020/02/03/business/earth-fare-grocery-stores-luckys-market/index.html

[2] Allen, Scotty. “Inside the RFID Stickers from a Chinese Cashier-Less Store.” *YouTube*, 21 June 2018. http://www.youtube.com/watch?v=0QKrHi-G9WQ

[3] 2018. “Supermarket Facts” *Food Marketing Institute*. https://www.fmi.org/our-research/supermarket-facts

[4] Wikipedia Contributors (2019). *Software architecture*. Wikipedia. https://en.wikipedia.org/wiki/Software_architecture

[5] Wikipedia Contributors (2019). *Blackboard system*. Wikipedia. https://en.wikipedia.org/wiki/Blackboard_system

[6] “Open Grocery Database Project.” *Grocery.com*, 24 Jan. 2017. https://www.grocery.com/open-grocery-database-project/

[7] “Name Generator” *Name-generator.com*. https://www.name-generator.org.uk/

[8] Haahr, M., 2020. “RANDOM.ORG - True Random Number Service.” *Random.org*. https://www.random.org

References 4 and 5 are used for Section 9.1 to assist in identifying the architectural styles used in this project.